
BAYESIAN HEURISTIC APPROACH TO DISCRETE AND GLOBAL OPTIMIZATION

Algorithms, visualization,
software, and applications

BAYESIAN HEURISTIC APPROACH TO DISCRETE AND GLOBAL OPTIMIZATION

Algorithms, visualization,
software, and applications

Jonas MOCKUS

*Institute of Mathematics and Informatics,
Kaunas Technological University,
Vytautas Magnus University, Vilnius Technical University
Akademijos 4, Vilnius 2600, Lithuania*



William EDDY

*Department of Statistics, Carnegie-Mellon University
Pittsburgh, PA 15217*



Audris MOCKUS

*Bell Laboratories
Naperville, IL 60566*



Linas MOCKUS and Gintaras REKLAITIS

*School of Chemical Engineering, Purdue University
W.Lafayette, IN 47907-1283*

KLUWER ACADEMIC PUBLISHERS
Boston/London/Dordrecht

CONTENTS

PREFACE	xiii
Part I BAYESIAN APPROACH	1
1 DIFFERENT APPROACHES TO NUMERICAL TECHNIQUES AND DIFFERENT WAYS OF REGARDING HEURISTICS: POSSIBILITIES AND LIMITATIONS	3
1.1 Outline	3
1.2 Sources of Heuristics	6
1.3 Different Approaches	10
1.4 Minimax Approach: Worst Case Analysis	14
1.5 Pareto-Optimal Approach: Dominant Analysis	18
1.6 Bayesian Approach: Average Case Analysis	19
1.7 Bayesian Heuristic Approach: Average Case Heuristic Analysis	21
1.8 Dynamic Visualization Approach: Interactive Analysis	25
1.9 List of Applications	26
1.10 Batch Process Scheduling	28
1.11 Global Optimization Software	29
2 INFORMATION-BASED COMPLEXITY (IBC) AND THE BAYESIAN HEURISTIC APPROACH	31
2.1 Introduction	31
2.2 Outline of IBC	31

2.3	IBC and Global Optimization	36
2.4	Bayesian Approach	38
2.5	A Posteriori Measure	39
2.6	Bayesian Heuristic Approach	40
2.7	Parallel Bayesian Algorithms	45
2.8	Conclusions	46
3	MATHEMATICAL JUSTIFICATION OF THE BAYESIAN HEURISTICS APPROACH	47
3.1	Introduction	47
3.2	Definition of Convergence	48
3.3	Advantages of Randomized Heuristics	53
3.4	"Learning" of BHA	56
Part II	GLOBAL OPTIMIZATION	61
4	BAYESIAN APPROACH TO CONTINUOUS GLOBAL AND STOCHASTIC OPTIMIZATION	63
4.1	Method of Search	63
4.2	Defining A Priori Distribution	64
4.3	Updating A Priori Distributions	65
4.4	Defining and Minimizing Risk Functions	66
4.5	Convergence of Bayesian Methods	67
5	EXAMPLES OF CONTINUOUS OPTIMIZATION	71
5.1	Outline	71
5.2	Maximization of the General Yield of Differential Amplifiers	72
5.3	Optimization of the Mechanical System of Shock-Absorber	74
5.4	Estimation of Non-linear Regression Parameters of the Im- munological Model	75
5.5	Estimation of Bilinear Time Series Parameters	76
5.6	Optimization of Composite laminates	76
5.7	The "Disk" Problem: Minimization of Potential Energy of the Organic Molecule Model	79

5.8	Planning of Extremal Experiments on Thermostable Poly- meric Composition	81
6	LONG-MEMORY PROCESSES AND EXCHANGE RATE FORECASTING	83
6.1	Introduction	83
6.2	Auto-Regression Fractionally-Integrated Moving-Average Mod- els (ARFIMA)	84
6.3	Artificial Neural Networks Models (ANN)	88
6.4	Bilinear Models	89
6.5	Variable Structure Models (VSM)	90
6.6	Optimization of ARFIMA Models	101
6.7	Optimization of ARMA Models	111
7	OPTIMIZATION PROBLEMS IN SIMPLE COMPETITIVE MODEL	119
7.1	Introduction	119
7.2	Competitive Model	120
7.3	Search for Equilibrium	120
7.4	"Social" Model	125
7.5	Lagrange Multipliers	126
Part III	NETWORKS OPTIMIZATION	129
8	APPLICATION OF GLOBAL LINE-SEARCH IN THE OPTIMIZATION OF NETWORKS	131
8.1	Introduction	131
8.2	Global Line-Search (GLS)	131
8.3	Optimization of Networks	132
8.4	Optimization of High-Voltage Power System Networks	134
8.5	Mixed Integer Line-Search	136
9	SOLVING DIFFERENTIAL EQUATIONS BY EVENT- DRIVEN TECHNIQUES FOR PARAMETER OPTIMIZATION	139
9.1	Introduction	139
9.2	Event-Driven Techniques	139

9.3	Critical Moment	141
9.4	Twin-Node Event-Driven Techniques	146
9.5	Computing Results	147
10	OPTIMIZATION IN NEURAL NETWORKS	153
10.1	Introduction	153
10.2	Error Surfaces	154
10.3	Perceptron and Linear Discriminant	155
10.4	Testing Highleyman's Problem	159
10.5	Error Surfaces of the Perceptron Unit	161
10.6	Composing a Network	164
10.7	Learning Weights Dynamics	166
10.8	Why Local Search Works?	171
Part IV	DISCRETE OPTIMIZATION	175
11	BAYESIAN APPROACH TO DISCRETE OPTIMIZATION	177
11.1	Introduction	177
11.2	Sequential Decision Problem of Discrete Optimization	177
11.3	Exact and Approximate Methods of Discrete Optimization	178
11.4	Definition of Randomized Heuristics	179
11.5	Algorithm of Randomized Heuristics	180
11.6	Reduction to Continuous Stochastic Optimization	189
11.7	Convergence	190
11.8	Improving Efficiency	190
11.9	Methods of Global Stochastic Optimization	192
11.10	Application to Stochastic Discrete Optimization	193
12	EXAMPLES OF DISCRETE OPTIMIZATION	195
12.1	Knapsack Problem	195
12.2	Travelling Salesman Problem	200
12.3	Flow-Shop Problem	204
12.4	Job-Shop Problem	215
12.5	Parameter Grouping	216
12.6	Conclusions	219

13 APPLICATION OF BHA TO MIXED INTEGER NONLINEAR PROGRAMMING (MINLP)	221
13.1 Introduction	221
13.2 Applying of Genetic Algorithms (GA) to MILP problem	222
13.3 Applying Penalty Function Heuristics (PFH) to Mixed Integer Bilinear Programming (MIBLP)	226
Part V BATCH PROCESS SCHEDULING	231
14 BATCH/SEMI-CONTINUOUS PROCESS SCHEDULING USING MRP HEURISTICS	233
14.1 Introduction	233
14.2 Different Models	235
14.3 Illustrative Example	237
14.4 Applying BHA	240
14.5 Bayesian Approach using MRP Heuristics	241
15 BATCH PROCESS SCHEDULING USING SIMULATED ANNEALING	245
15.1 Introduction	245
15.2 Outline	245
15.3 Model Description	246
15.4 Formulation Example	249
15.5 Computing Results	254
15.6 Analysis of Results	259
16 GENETIC ALGORITHMS FOR BATCH PROCESS SCHEDULING USING BHA AND MILP FORMULATION	261
16.1 Introduction	261
16.2 Equalities and Inequalities	261
16.3 Objective Function	264
16.4 Application of BHA to the Batch Scheduling Problem, using MILP and GA	265
16.5 Aggregate of Batch Scheduling Problems	268

Part VI SOFTWARE FOR GLOBAL OPTIMIZATION	275
17 INTRODUCTION TO GLOBAL OPTIMIZATION SOFTWARE (GM)	277
17.1 Background	277
17.2 Typical Examples	279
17.3 Different Versions	280
18 PORTABLE FORTRAN LIBRARY FOR CONTINUOUS GLOBAL OPTIMIZATION	283
18.1 Introduction	283
18.2 Preliminary Discussion	285
18.3 Description of Routines	291
19 SOFTWARE FOR CONTINUOUS GLOBAL OPTIMIZATION USING UNIX C++	327
19.1 User's Reference	327
19.2 Description of Methods	331
19.3 HP-UNIX Class-Room Version of GM	333
20 EXAMPLES OF UNIX C++ SOFTWARE APPLICATIONS	337
20.1 Introduction	337
20.2 Exchange Rates Long-Term Prediction Using the ARFIMA Model and GM in Interactive Mode	338
20.3 Exchange Rates "One-Step" Prediction Using the ARMA Model and GM in Batch Mode	343
20.4 Optimizing BHA Parameters in Flow-Shop Problem	345
20.5 Optimizing BHA Parameters in Knapsack Problem	345
Part VII VISUALIZATION	347
21 DYNAMIC VISUALIZATION IN MODELING AND OPTIMIZATION OF ILL DEFINED PROBLEMS: CASE STUDIES AND GENERALIZATIONS	349

CONTENTS

xi

21.1 Introduction	349
21.2 Interactive and Non-interactive Dynamic Graphics	351
21.3 Dynamic Graphics in the Manufacturing Process	354
21.4 Non-interactive Dynamic Maps in the Epidemiological Model	358
21.5 Interactive Dynamic Maps	364
21.6 Interactive Icon Index	369
21.7 Summary	377

REFERENCES	379
-------------------	-----

INDEX	393
--------------	-----

PREFACE

The objective of this book is to discuss in one volume different approaches and different heuristic techniques of optimization. We proceed from the formal Bayesian Approach (BA) to the semi-formal Bayesian Heuristics Approach (BHA) and to the informal Dynamic Visualization Approach (DVA). We show the advantages and disadvantages of BA, BHA, and DVA, applying these approaches to different problems of global and discrete optimization.

The authors attempt to provide a setting in which one can discuss a Bayesian adaptive choice of heuristics for discrete and global optimization problems. We make predictions taking into account partial information and evaluate all in the spirit of the average rather than the worst case analysis. The role of visualization is also considered.

In this context "heuristics" are understood to be an expert opinion defining how to solve a family of problems of discrete or global optimization. The main goal is to describe in one volume different ways of applying heuristics. Different ways represent different degrees of formalization. We start from the traditional Bayesian Approach (BA), where heuristics are included by a choice of an a priori distribution. We extend the formal BA to a semi-formal Bayesian Heuristic Approach (BHA) where heuristics may be included more flexibly. We finish by describing the Dynamic Visualization Approach (DVA). Using DVA, we can include heuristics directly, bypassing any formal mathematical framework.

All theoretical results developed in this book are applied in real-life or test problems. The application examples illustrate theoretical results. One of the real-life families of application examples, scheduling of batch operations, is described in detail because this family illustrates most of the theoretical results.

The theoretical discussions are for the readers with a mathematical background. However, one may skip the theoretical part of the book if only the applications are of interest.

In the introductory Part I, Bayesian Approach, the results are discussed at several levels in order to facilitate understanding of what the book is about. At the first level (Chapter 1), only the general ideas are outlined. At the second level (Chapter 2), the Bayesian Heuristic Approach is discussed using the terms of the well known complexity theory. In Chapter 3, mathematical justification of BHA is considered.

In Part II, Global Optimization, the theory and some application examples of the Bayesian Approach to continuous global optimization are described.

In Part III, Network Optimization, useful heuristics are considered and the power of BHA is discussed for network problems.

In Part IV, Discrete Optimization, we show how to reduce discrete optimization problems to continuous stochastic ones, using the Bayesian Heuristic Approach. The traditional knapsack, travelling salesman, and flow-shop problems are considered as test problems. The application of BHA to Mixed Integer Nonlinear Programming (MINLP) is discussed.

In Part V, Batch Process Scheduling, the real scheduling problem is considered as an example of BHA application, using various heuristics.

In Part VI, Software for Global Optimization, the global optimization software is described for both UNIX and MS-DOS environments.

In Part VII, Visualization, the Dynamic Visualization Approach is considered as a promising technique for solving ill-defined optimization problems.

The enclosed disk contains a LINUX version of the UNIX (X-windows) global optimization software (GM), including application examples. A portable Fortran library is also included for the users (see README in the disc). The GM software is intended for "active" readers, who would like to apply the results to their own problems immediately. Contact e-mail addresses:

jonas.mockus@ktl.mii.lt

jonas@optimum.mii.lt

audris@bell-labs.com

mockus@ecn.purdue.edu

The theoretical aspects of BHA and the book editing was performed by Jonas Mockus, the numerical results are due to Linas Mockus and Audris Mockus, the

Batch Scheduling part is contributed by G. Reklaitis and L. Mockus, and the Dynamic Visualization part by W. Eddy and A. Mockus.

Finally, we wish to express our thanks to colleagues who directly or indirectly contributed in the process of writing this book. We further acknowledge the support of the publisher in producing the book.

J. Mockus, W. Eddy, A. Mockus, L. Mockus, and G. Reklaitis

July 1996

PART I

BAYESIAN APPROACH

DIFFERENT APPROACHES TO NUMERICAL TECHNIQUES AND DIFFERENT WAYS OF REGARDING HEURISTICS: POSSIBILITIES AND LIMITATIONS

1.1 OUTLINE

1.1.1 Objective

The aim of this book is to investigate the advantages and limitations of different approaches and different heuristic techniques of optimization. We proceed from the formal Bayesian Approach (BA) to the semi-formal Bayesian Heuristic Approach (BHA) and finally, to the informal Dynamic Visualization Approach (DVA). Therefore, in addition to BA and BHA algorithms, we discuss various visualization techniques and their application to case studies of optimal decision making.

We start the description from the application of BA in the continuous global optimization. Then we show how to extend the results to the BHA, namely, to the optimization of parameters of randomized heuristic techniques of discrete and global optimization.

A new concept employed in this book is to define an a priori distribution on a set of parameters of randomized heuristics. The traditional way is to define it on a set of functions to be minimized. The definition of the a priori distribution on the set of heuristic decision rules is a vehicle for including expert knowledge more flexibly and a means of speeding up the search.

We show the advantages and disadvantages of BA and BHA applying those approaches to different problems of global and discrete optimization.

We describe the software for UNIX and DOS platforms and enclose the disk.

1.1.2 Different Formalization Degrees

The maximal deviation is traditionally used while developing various numerical techniques. We define that as the Minimax Approach (MMA) or the worst case analysis.

The advantage of MMA is a complete formalization. Everything is well defined, and one obtains results with a guarantee. A disadvantage is the high price we have to pay for the guarantee. By guarantee we mean maintaining a fixed error limit for all the functions to be optimized. The guarantee price often is an algorithm of exponential time, that is, the time of solution exponentially depends on the problem complexity (see Sub-section 2.2.5).

In this book we consider the average deviation as a criterion when designing numerical optimization techniques and algorithms. We define that as the Bayesian Approach (BA).

A disadvantage of BA is the degree of uncertainty while defining an a priori distribution on the set of problems to be optimized. Thus one obtains no performance guarantee. An advantage is the possibility to include some elements of expert knowledge while defining an a priori distribution. By involving expert knowledge one can tailor the algorithm to specific problems. In this way the efficiency of algorithms is increased.

In some cases it is more convenient or efficient to express expert knowledge in the form of a priori distribution on a set of heuristics rather than on a set of problems. People often become experts by acquiring knowledge in a domain while applying different solution methods to encountered problems. Hence, such expert knows which methods worked well and which did not. This type of experience is most naturally incorporated into BHA framework. We refer to this extension of traditional BA as a Bayesian Heuristic Approach (BHA). Thus we obtain less formalization but more flexibility.

Both BA and BHA help the algorithm developers to include the expert knowledge into a regular mathematical framework. Dynamic visualization helps the decision makers to include expert knowledge directly in cases when the objective function has multiple criteria (is infinite-dimensional) or is not well defined. The expert knowledge is used to define visual-time representation of the object-

ive and the optimization stage involves visual inspection of the modeling results under various parameters with the goal to reach qualitative decisions. We define this as the Dynamic Visualization Approach (DVA)

1.1.3 Heuristics Randomization

Traditionally heuristics are understood to be a set of rules defining decision priorities based on the experience and intuition of experts [111]. The priorities are defined in two ways:

- by the decision rules ordering the decisions,
- by the utility function on the decision set.

The first way defines which decision is better but does not say how much better. That is acceptable, if one uses only "pure" decisions. However, we have to know how much better, if we desire to lift this restriction and want to evaluate the "mixed" decisions. The advantages of mixed strategies in games are well known [112]. In game theory, mixed strategies denote the probabilities of discrete decisions. Using mixed strategies one extends the set of discrete decisions to a convex set, thus providing equilibrium conditions (see [112] and Chapter 7). In Section 3.3 we illustrate the advantages of the mixed heuristic decisions. Mixing the heuristic decisions by randomization we extend the discrete decision set to the continuous one. Consequently, we obtain better average results by optimizing on the extended decision set.

The BHA in discrete optimization is similar to the search for equilibrium solutions in games. In game theory, we optimize the mixed strategies to obtain the equilibrium solution to the game. Using BHA we optimize the parameters of randomized heuristics to adapt the heuristics to the original optimization problem or to the family of optimization problems. Defining BHA we assume randomization parameters such that convergence is attained (see Section 3.2).

1.1.4 Heuristics Optimization

Many authors, who apply randomized heuristics, adapt the heuristic decision parameters "by hand" as a part of the research process. This adaptation may be formalized as an optimization problem. We regard the randomization parameters as real numbers. The outcome of optimization is stochastic. The function

to be optimized is multi-modal in many cases. Thus the optimization of randomization parameters, in general, is a problem of continuous stochastic global optimization. We may optimize the parameters by any suitable method, not necessarily by BA. However, we prefer to use the BA for both theoretical and practical reasons

- theoretically BHA is interesting because it unites the Bayesian Approach and Heuristic Programming [110, 111] by defining an a priori distribution on a family of heuristics.
- the practical reason is that the Bayesian methods [100] are designed for global stochastic optimization (see Chapter 4) so we may expect good results in optimizing the multi-modal stochastic auxiliary function that defines how the best decision depends on the randomization parameters.

1.2 SOURCES OF HEURISTICS

1.2.1 Classification

A rigorous formal definition of heuristics would be difficult and not very useful because strict formalisms contradict the informal nature of heuristic knowledge [111]. Thus we introduce the following informal classification of different types of heuristics:

- "greedy" heuristics, those which "build" a system by adding the elements, that are the "most desirable" [59] based on local information (see Section 11.4);
- "permutation" heuristics, those which start from a feasible solution, and decide which of several permutations to accept based on local information (see Section 11.4);
- "deterministic search" heuristics, those which employ some well-known deterministic approximate search algorithm, for example, the Gupta heuristics (12.11) in the flow-shop problem;
- "stochastic search" heuristics, those which consider a stochastic search algorithm as a heuristic, for example, the Simulated Annealing algorithm (see Sub-section 11.5.4).

In all the cases we "improve" the heuristics by randomization and by optimizing the randomization parameters. A possible exception is the "stochastic search" case in which we may optimize parameters of the algorithm directly because the algorithm is stochastic by its definition.

1.2.2 Greedy and Permutation Heuristics

"Greedy" heuristics are an important family of heuristics. In [59] the following definition is given "a greedy algorithm, when run on a set system, builds a solution by beginning with the empty set and successively adding the best remaining element while maintaining feasibility." Here the number of decision stages I is equal to the number of the system elements.

Another family of heuristics are so-called "permutation" heuristics. Using this strategy one starts from some initial state of the system and then performs a number of permutations. One stops, if using the given set of permutations cannot improve the results. In such a case, the number of stages may be much larger than the number of system elements.

Let us explain the difference between greedy and permutation heuristics by using the example of the well known travelling salesman problem. The simplest technique is to connect the cities sequentially, starting the path of a salesman from some initial city. A greedy heuristic would be to focus on the distance (with a negative sign¹) to some non-connected city. A pure heuristic decision would be to go to the nearest non-connected city. A randomized heuristic would be to go to a non-connected city with a probability that depends on the distance.

A permutation way to define heuristics is to start from a certain initial path and define feasible permutations of that path as follows. First one selects two pairs of adjacent cities $A - B$ and $C - D$ on this path. A feasible choice would be one in which we break the connections $A - B$ and $C - D$, and reconnect A to C and B to D while still obtaining a path that visits all cities in succession.

Thus by fixing a pair $C - D$ and reconnecting A to C and B to D we define a permutation. In this case the permutation heuristics for each given pair $A - B$ is the length of the reconnected path (with a negative sign) that corresponds to the pair $C - D$. So a pure heuristics is to select a pair $C - D$ corresponding to the shorter reconnected path. A randomized heuristics is to select a pair $C - D$ with a probability that depends on the length of reconnected path.

¹We regard greater heuristics as better ones.

An advantage of permutation heuristics is the wide choice of different feasible permutations. Another advantage is that we can use expert knowledge by selecting a favorable initial solution.

If no initial solution is available, then one can optimize in two stages. In the first stage we use a greedy algorithm which starts from the empty set and constructs a feasible solution. In the second stage one improves the "greedy" solution by a feasible permutation. It is possible to eliminate the first stage of the optimization, if there is a good expert who can provide the initial solution.

1.2.3 Stochastic Search Heuristics

Simulated Annealing

A popular method of global optimization is simulated annealing, which may be considered in the BHA framework using heuristics defined in Sub-section 11.5.4. The main difference from the traditional simulating annealing is that using BHA we optimize the "initial temperature" for some fixed number of iterations.

Genetic Algorithms (GA), GRASP system, and "Taboo" Search

Genetic Algorithms [4] are an important "source" of interesting and useful stochastic search heuristics. It is well known that the results of the genetic algorithms depend on the mutation and cross-over parameters (see Chapter 16 and Section 13.2). The Bayesian Heuristic Approach could be used in optimizing those parameters.

In a GRASP system (see [42, 114, 130, 116, 117, 131]) the heuristic is repeated many times. During each iteration a greedy randomized solution is constructed and the neighborhood around that solution is searched for a local optimum.

The "greedy" component constructs a solution, one element at a time (in the style of a greedy heuristic), by doing the following:

- apply to all not yet unselected candidates a GREEDY function;
- sort these candidates according to the GREEDY function;

- select a subset, consisting of GOOD (but not only the BEST) candidates to be included in the Restricted Candidate List (RCL);
- select a CANDIDATE, at RANDOM, from RCL to be in the solution;
- change the GREEDY function to take into account the inclusion of the candidate into the solution.

These steps are repeated until a solution is constructed.

A possible application of the BHA in GRASP is in optimizing a random selection of a candidate to be in solution because different random selection rules could be used and their best parameters should be defined. BHA might be useful as a local component, too, by randomizing the local decisions and optimizing the corresponding parameters.

In "Taboo" Search [115] some parameters also need to be optimized. It seems the Bayesian Heuristics Approach may be considered when applying almost any stochastic or approximate method of discrete optimization. The proven convergence of a discrete search method (see, for example, [3]) is an asset. If convergence is proven, then no additional testing of convergence conditions (see Theorem 3.2.1) is needed.

1.2.4 Deterministic Search Heuristics

Lower Bounds and Penalty Functions

Lower bounds² (see Chapter 13) may be regarded as useful heuristics, if feasible solutions may be easily obtained. Using BHA one defines the probability $r(m)$ of decision m as a function of the lower bound $h(m)$ corresponding to this decision. The lower bound techniques are developed in the Branch-and-Bound (B&B) algorithms. If insuring feasibility is not easy, then some form of penalty function may be accepted as a heuristic (see Chapter 13).

Heuristic Generalization

One may often obtain good average results by applying to more general cases algorithms that provide exact solutions for special cases. An example is the

²Upper bounds are considered if we maximize the objective function.

Gupta heuristic in the flow-shop problem ³ (see Chapter 11). Another example is the Global Line Search (GLS) in the high-voltage electrical network optimization problem (see Chapter 8). In this case, the GLS algorithm yields the exact solution in both the linear and the zero-one case. This algorithm may be regarded as a good heuristic in the cases which lie "between" the continuous linear and the strictly zero-one.

A popular example of heuristic generalization is the application of local search while solving multi-modal problems. In Chapter 10, we consider why local search works well in the multi-modal problem of Artificial Neural Networks (ANN) optimization. Here the heuristics are involved in choosing proper initial points. One starts from the solution of linear problem corresponding to small inputs. Afterwards the linear solution is improved, by gradually introducing non-linearities related to greater inputs. In Chapter 10, only the deterministic case of ANN optimization is considered. However, we expect that the efficiency of search will be increased using BHA by introducing randomization and optimizing the randomization parameters.

1.3 DIFFERENT APPROACHES

1.3.1 Minimax Approach

Mathematical optimization methods traditionally are evaluated by their maximal error. We define that as the worst case analysis or a Minimax Approach (MMA). A practical advantage of MMA is that we guarantee that the error will not exceed some ϵ -limit. A theoretical advantage of MMA is that no subjective judgments or uncertain data are required. Unfortunately, the price of the guarantee and the theoretical elegance may be too high.

For example, the usual price of a guarantee while solving the NP-complete problems is an exponential time algorithm (see [79] and Sub-section 2.2.5). This means that when one starts designing some MMA algorithm one has to answer the following questions:

- is the problem NP-complete?
- can we "afford" an exponential time algorithm?

³The Gupta heuristic provides the exact solution to the two-machine flow-shop problem

In solving real-life optimization problems one often cannot afford exponential time. Thus, we replace the guarantee by some more relaxed conditions. We use not only the well-defined specific properties of the given problem, but also informal expert knowledge. The subjective components of expert knowledge are referred to as heuristic knowledge or simply "heuristics".

An a priori distribution is a traditional way of representing the subjective knowledge in the framework of statistical decision theory (see [27, 28]). One determines the average error of a given optimization algorithm by defining an a priori distribution on a set of objective functions. Consequently, one designs optimization methods which minimize the average error. We define that as a Bayesian Approach (BA). Clearly BA may be regarded as an indirect way of using heuristics through the vehicle of the a priori distribution. That is a well formalized but rather inflexible way.

It is of interest to find other more flexible ways of including heuristics within the Bayesian framework. We define that as a Bayesian Heuristic Approach. In the following we will consider this approach in a step-by-step fashion.

1.3.2 Heuristic Approach (HA)

As the first step, let us define a pure Heuristic Approach (HA). We represent the heuristics as some function h_i , where i denotes the decision index. HA involves maximization of the heuristics h_i at each iteration. In some cases, by maximizing the heuristics we may obtain the exact solution [59]. However, in most cases, heuristic algorithms stop short of reaching the exact solution. This means one cannot improve the pure heuristic solution any more even in case one has available a large amount of computer power.

The advantage is that by applying HA we may decrease the average error, depending on the "quality" of heuristic" h_i . An important problem arises how to estimate this quality and how to improve it.

1.3.3 Randomization Approach (RA)

As the second step, we consider a Randomization Approach (RA). We take a decision i with some probability r_i . The advantage of RA is that we obtain the optimal discrete solution with probability 1, if the minimal r_i is positive (see Theorem 3.2.1). However, one may need the exponential time, if r_i is generated

more or less uniformly while solving an NP-complete problem (see Sub-section 2.2.5). A uniform distribution in RA means that we do not apply the expert knowledge.

1.3.4 Randomized Heuristic Approach (RHA)

As the third step, to derive the advantages of both HA and RA, we consider a Randomized Heuristic Approach (RHA). We define probabilities r_i as some function $r(h_i)$ of heuristics h_i . A simple choice of $r(h_i)$ is a linear function of h_i .

One may improve the results of RHA by considering not a single function $r(h_i)$ but a "mixture" of m different functions

$$r^1(h_i), r^2(h_i), \dots, r^m(h_i).$$

One encounters an additional decision problem: which of these functions to use. We may make the choice by a "lottery":

$$x = (x^1, x^2, \dots, x^m)$$

where x^j denotes the probability of "winning" the randomization function r^j .

Denote by $f(x) = f_K(x)$ the best result obtained by applying RHA K times to some optimization problem $\min_d v(d)$, where v is an objective function (of the original optimization problem) and d is the decision d .

In this book, as usual, we denote a function of continuous variables x by $f(x)$. By $v(d)$ we denote a function of variables d which might be discrete. Such a notation is convenient using BHA, since the auxiliary function $f(x)$ depends on continuous "winning" probabilities x . The original objective function $v(d)$ may depend on discrete or continuous variables d defining the parameters of the original problem.

If $\min_i r_i > 0$, then

$$\lim_{K \rightarrow \infty} f_K(x) = \min_d v(d). \quad (1.1)$$

Expression (1.1) follows from Theorem 3.2.1.

1.3.5 Bayesian Heuristic Approach (BHA)

We see that $f(x)$ is, in general, a multi-modal stochastic function. Thus, a natural way of minimizing it is by defining an a priori distribution on the set of randomization parameters x . This is the last step in defining the Bayesian Heuristic Approach. We regard BHA as a semi-formal way of including heuristics into a mathematical framework.

BHA uses heuristics in two ways:

- directly, as the arguments of randomization functions $r^j(h_i)$;
- indirectly, by the a priori distribution on the set of randomization parameters x .

An additional advantage of BHA is its "learning" ability. By learning we mean the possibility to apply the optimal randomization parameters x obtained while solving problems from the "learning" set to other related problems. Learning is important in the "on-line" or real-time mode of optimization, when there are strict computing time limits.

1.3.6 Dynamic Visualization Approach (DVA)

We started our classification of algorithms from a strictly formal MMA. Then we considered a traditional BA and a semi-formal BHA. An informal interactive optimization is needed if an optimization problem is not well defined. This may arise, for example, if the mathematical model, including the objective function, must be updated during the course of optimization process.

The informal interactive approach attempts to represent an optimization problem in a visual form that is domain specific and is intuitive to the domain

expert. The visual representations can vary significantly across the domains. In the examples the dynamic visual representation of a smooth function in time and space turned out to be effective in several domains. The domain specific visual representation can efficiently convey information about a complex model and help make qualitative judgments about model's adequacy and optimality.

The efficiency of informal interactive optimization depends on dynamic visualization techniques. We regard dynamic visualization as an important tool using heuristics in an informal interactive way and thus will refer to it as a Dynamic Visualization Approach. Two basic techniques of dynamic visualization are considered:

- space-time smoothing;
- image search.

Those techniques will be explained through real life examples. Now we shall consider these and some other approaches in more detail.

1.4 MINIMAX APPROACH: WORST CASE ANALYSIS

A traditional approach to numerical methods is to design a sequence of points

$$x_n \in A \subset R^m, \quad n = 1, 2, \dots$$

such that converges to an exact solution x^* for all problems from a given family, when n is large. In some simple cases, usually connected with convexity, the convergence rate can also be defined. In terms of decision theory this approach can be considered as the "Worst Case Analysis", or a "Minimax Approach". This means that the method must retain some property in all cases, including the worst one.

This approach seems so natural that numerical analysts usually assume it to be the only rigorously acceptable one. Other approaches are usually considered to be "heuristic." Heuristic here means methods that may be practically quite efficient, but without proper mathematical justification. The mathematical justification is frequently assumed to be the necessary part of a serious numerical analysis. So any property of numerical methods that does not hold for all the

problems from the given family is often regarded as some empirical evidence at best.

An obvious advantage of the traditional approach is that it helps to maintain strict standards in the numerical analysis. It does not allow a flooding of the field by numerical methods with unknown mathematical properties.

An important disadvantage is that the Minimax Approach is too expensive, in general. To obtain the exact solution in the worst case one needs many iterations, if the family of problems is large enough. In a large family of problems the worst case can be expected to be very bad indeed.

Let us consider, for example, the global optimization of a family of Lipschitz functions with an unknown Lipschitz constant. In this case, the best method, in a minimax sense, is a uniform grid on a compact feasible set [143]. This means that the global optimization algorithm will be of exponential time⁴. The number of required observations will be increasing exponentially with the complexity of the problem. We define complexity as the number of variables and the accuracy of solution (see Sub-section 2.2.5). The term "observation" denotes an evaluation of the objective function $f(x)$ at some fixed point x .

If the Lipschitz constant is known, then some nonuniform grid technique is preferable, [40]. However, even here the time of the algorithm, apparently, remains exponential, although with a better rate parameter.

We cannot apply the Minimax Approach to the global optimization of continuous functions at all. The maximum does not exist on the set of all continuous functions because this set is not a compact one. This means that for any fixed continuous function and a fixed method of search there exists another continuous function with a larger deviation from the global minimum. Therefore, the condition of uniform convergence does not apply here.

Some weaker convergence conditions are usually considered, for example, the condition of convergence for any fixed continuous function. However, to satisfy even this, much weaker, condition, one needs an exponential time algorithm (see [79]). The convergence for any continuous function can be achieved only by asymptotically dense observations. This means that the maximal distance between observations converges to zero. Otherwise, we can miss the global minimum for some continuous function.

⁴The well known and rather efficient algorithm for Lipschitzian optimization without the Lipschitz constant (see [73]) is not a minimax one.

The problem becomes even more complicated in the presence of "noise". For example, the noise is present if we define an objective function by Monte Carlo techniques or by physical experiments.

The convergence of the best observation (the so-called "hit-convergence" (see Section 3.2)) to the global optimum is rather weak. It holds for any global optimization method that provides asymptotically dense observations of continuous functions. This means that this convergence can be regarded only as a necessary condition. Some additional conditions should be included if we wish to justify our method sufficiently.

Following the traditions of local optimization, one would like to prove not only convergence but also the rate of convergence. However, we cannot do that for the set of all continuous functions. This set is too large. The trouble is that usually we define the convergence rate using a notion of supremum directly or indirectly. This notion does not apply to a non-compact set of all continuous functions. Thus, different ways to make the convergence conditions stronger must be investigated.

One way is to consider a density ratio instead of the rate of convergence. We define the density ratio as a ratio of density of observations in a vicinity of global minimum to the average density of observations (see Section 3.2). This defines the asymptotic efficiency of the methods of global optimization reasonably well. Therefore, we regard the density ratio in the global optimization using BA as a replacement for the traditional rate of convergence.

Another way is to regard the convergence of the last observation, the so called "stay-convergence" (see Section 3.2). This convergence is usually considered in simulated annealing algorithms [154]. Convergence and even convergence rate are inherently asymptotic properties of numerical algorithms. However a useful method should not only have good asymptotic behavior but also exhibit effective performance over a finite number of observations. This means that good asymptotic behavior can be regarded as a necessary but not sufficient condition. To justify any claim for practical efficiency of a method of global optimization some additional "non-asymptotic" conditions should be considered. It is well known that in real life applications even the best asymptotic property can happen to be nearly useless if the number of observations is not large enough or if the observations lie outside of the region of convergence.

Under the "worst case" notion we define the "optimal" method as a method that provides a minimal or at least a reasonably low deviation from the global optimum for all functions of the given family. This of necessity requires definition

of what constitute the "worst case". Such a definition is not possible if the set of functions is not compact. This in fact is the major theoretical difficulty of the "worst case" approach. The practical difficulty is that the worst case can be very bad, if the family of functions is sufficiently large.

For example, for smooth convex functions the well known variable metric methods are, perhaps, nearly optimal. A super-linear convergence was proved [124] for those methods. For the family of one-dimensional unimodal functions [77] quite an efficient optimal method in the minimax sense was developed. Some optimal methods, in the minimax sense, are available for the set of Lipschitz functions [40, 123, 138, 143].

Some of these objective functions belong to a limited set of functions, namely, smooth convex or one-dimensional uni-modal. It is easy to see from these examples that the minimax (or approximately minimax) methods can be regarded as nearly optimal from both the theoretical and the practical point of view. For a wider family of functions, such as Lipschitz functions with an unknown constant, the minimax approach is not so attractive. The guarantee of optimality of the method is too expensive in the problems of high complexity in the sense of high dimension and accuracy (see Section 2.2.5).

For the family of continuous functions as well as as for functions with noise the worst case does not exist. Therefore, the worst case analysis is impossible. In the global optimization of continuous functions, the average case analysis seems to be a reasonable way to make the conditions of practical efficiency and mathematical justification compatible.

For a theoretical justification of the numerical method, we need to prove some good properties of the method under well defined conditions. For example, assuming continuity, differentiability, or the existence of the Lipschitz constant of the objective functions, homogeneity and the independence of the m -th differences of the a priori distribution (see [100]) and so on. A formal test of these conditions in real life applications is a problem nearly as complex as that of global optimization. Thus the correspondence of applied problems to the theoretical conditions is judged by intuition of experts.

1.5 PARETO-OPTIMAL APPROACH: DOMINANT ANALYSIS

The concept of Pareto Optimality (PO) (see [114]) is traditionally used regarding the cases when an objective is a vector-function $f_\omega(x)$, $\omega \in \Omega$, where $x \in A \subset R^m$ is the control parameter, ω is a component index of the vector-objective $f_\omega(x)$, and Ω is a set of all indices ω . A decision x^* is called Pareto Optimal⁵ if there is no dominant decision x such that

$$\begin{aligned} f_\omega(x) &\leq f_\omega(x^*), \text{ for all } \omega \in \Omega \\ f_\omega(x) &< f_\omega(x^*), \text{ for at least one } \omega \in \Omega. \end{aligned} \quad (1.2)$$

In this section we talk about a non-traditional application of Pareto Optimality while designing algorithms for optimization of *scalar* objective functions.

We explain the application of the PO concept in optimization of a scalar objective function using an algorithm for Lipschitzian optimization without the Lipschitz constant called DIRECT (see [74]) as an example⁶. In this example Ω denotes a set of all Lipschitz functions with unknown Lipschitz constants ω . We consider the DIRECT algorithm [74] in one dimension as an illustration. This algorithm partitions the space into intervals whose center points are evaluated. The basic idea of DIRECT is to select (and sample within) all "potentially optimal" intervals during an iteration. A formal definition of potentially optimal intervals follows.

Suppose that we have partitioned the interval $[a, b]$ into intervals $[a_i, b_i]$ with midpoints c_i , for $i = 1, \dots, m$. Let $\epsilon > 0$ be a positive constant, and f_{min} be the current best value. Interval j is said to be potentially optimal if there exists some rate-of-change constant $\omega > 0$ such that

$$\begin{aligned} f(c_j) - \omega (b_j - a_j)/2 &\leq f(c_i) - \omega (b_i - a_i)/2 \text{ for all } i = 1, \dots, m \\ f(c_j) - \omega (b_j - a_j)/2 &\leq f_{min} - \epsilon |f_{min}|. \end{aligned} \quad (1.3)$$

Numerical results in [74] have shown that this method has low computational overhead and is efficient for bound-constrained, black-box global optimization

⁵Here we consider minimization, in maximization the inequalities should be reversed

⁶The cited authors describe their method making no reference to Pareto Optimality.

problems of small size. A constrained version of the algorithm has been in use at General Motors since 1993 for solving engineering design problems.

We extend the set of potentially optimal intervals to a set of Pareto Optimal (PO) intervals replacing condition (1.3) by the following condition of Pareto Optimality considering the same lower bounds as in condition (1.3). An interval j is called Pareto Optimal if there is no dominant interval i such that

$$\begin{aligned} f(c_i) - \omega (b_i - a_i)/2 &\leq f(c_j) - \omega (b_j - a_j)/2 \text{ for all } \omega \in \Omega \\ f(c_i) - \omega (b_i - a_i)/2 &< f(c_j) - \omega (b_j - a_j)/2 \\ &\text{for at least one } \omega \in \Omega. \end{aligned} \tag{1.4}$$

Note that upper inequalities (1.3) defining the set of potentially optimal intervals and upper inequalities (1.4) defining the PO intervals are the same. Only the lower inequalities differ. The PO "extension" (1.4) of the DIRECT algorithm (1.3) is important conceptually because it includes the specific DIRECT algorithm within a general PO framework. We do not investigate computational advantages and disadvantages of the PO version of DIRECT because the objective of this book is application of the Bayesian approach. We mentioned the Pareto-Optimal Approach merely as an important alternative.

1.6 BAYESIAN APPROACH: AVERAGE CASE ANALYSIS

A theoretical question related to the Average Case Analysis is how to define the notion of "average." Mathematically, an average is an integral. It is well known that to define an integral, some measure must be fixed, a convenient one being a probability measure P . This measure is a part of the problem definition. Therefore, P should be fixed before starting any investigation of the problem. In statistical decision theory [27], the measure P is called an a priori distribution. In this fashion we define the Bayesian Approach (BA) or the Average Case Analysis.

The first problem of the Bayesian Approach is how to define the a priori distribution P . The second one is how to update it using the results of observations $z_n = (x_i, f(x_i), i = 1, \dots, n), n = 1, \dots, N$. We define the updated distribution as an a posteriori distribution $P(z_n)$. The third problem of BA is how to

minimize the a posteriori risk function. The risk function $R_n(x)$ is an expected deviation from the global minimum at a fixed point x . The expectation is defined by the a posteriori distribution $P(z_n)$. The minimization of the risk function $R_n(x)$ serves to determine the point of the next observation x_{n+1} .

Since any Bayesian method depends on an a priori distribution by definition, it is desirable to define this distribution on the basis of some clear and simple assumptions. For example, it follows from the conditions of continuity of $f(x)$, homogeneity of P , and independence of the m -th differences that the a priori distribution P is Gaussian with a special covariance matrix (4.4) (see Section 4.2).

We update an a priori distribution by the well known formula of conditional probability. Unfortunately, to update a Gaussian distribution, one should invert the covariance matrix of the n -th order ⁷. Inverting is not practical if n is more than, say, 500. Here n is the number of observations. Since the covariance matrix represents Kolmogorov's consistency conditions (see [100]), the inversion can be avoided only by replacing the consistency conditions with something weaker.

Let us replace them by the following three conditions: continuity of the risk function $R_n(x)$, convergence of the Bayesian method to the global minimum of any continuous function $f(x)$, and "simplicity" of expressions defining a "conditional" expectation and a "conditional" variance (see Section 4.4). Thus, we define some "Bayesian" method that can be regarded as the simplest one under some assumptions. The term "Bayesian" here has a meaning different from the traditional definition of the Bayesian Approach. The reason is that a modified definition of "conditional" expectation and variance does not correspond to Kolmogorov's consistency conditions.

There are other ways of simplifying the expressions of conditional expectation and conditional variance. For example, [167] approximately expressed them using extrapolation theory.

A Bayesian algorithm converges to a global minimum of any continuous function, if the a priori distribution is chosen correctly [100]. This means that asymptotically the Bayesian method is at least as good as any traditional method which offers hit-convergence (see Section 3.2) for the family of continuous functions. In fact, it is usually even better because the asymptotic density of ob-

⁷In Markovian cases there exist simple ways of determining conditional probabilities. Unfortunately, we are able to define only one-dimensional Markovian functions.

servations of Bayesian methods is considerably higher near the global minimum (see Section 4.5). However, the main advantage of Bayesian methods is that they minimize an expected deviation from the global minimum for any fixed number of observations. It should be emphasized that the objective of Bayesian methods is different from the traditional asymptotic notions, including such definitions as the convergence rate and the exponential or the polynomial time algorithms (see Sub-section 2.2.5).

The problem of minimizing the risk function $R_n(x)$ is multi-modal even in the simplest case (see Section 4.4). Therefore, by using the Bayesian method we replace the original multi-modal problem by an auxiliary multi-modal problem. The advantage of doing so is that in optimizing the auxiliary problem even large deviations are not essential. We need the auxiliary problem merely for some rough prediction of the risk. Since one uses this prediction only to choose the best point for the next observation there is no need to minimize the risk function exactly. We can use simple methods such as Monte Carlo, to minimize $R_n(x)$ approximately. However, the optimization of a multi-modal risk function is itself a computationally demanding task. Therefore, the applications of Bayesian algorithms can be efficient only in the global optimization of "expensive" objective functions. The function $f(x)$ can be regarded as expensive if at least several minutes of CPU time are necessary for its evaluation. For inexpensive functions, simpler global optimization methods such as clustering [148], global line search (see chapters 18 and 19), uniform deterministic search (see chapters 18 and 19), and even the simplest Monte Carlo search could be more efficient.

It is well known that the intuition of experts depends on their practical experience. This means that the theoretical analysis of the methods of global optimization should be supplemented by the analysis of case studies, covering a sufficiently large family of different applied problems. It is done in the parts, chapters, and sections of the book oriented towards applications, considering several real life examples from very different fields.

1.7 BAYESIAN HEURISTIC APPROACH: AVERAGE CASE HEURISTIC ANALYSIS

Generally algorithms of exponential time are needed (see Sub-section 2.2.5) in order to obtain the exact solution of global and discrete optimization problems.

Screening techniques, such as Branch-and-Bound (B&B) can be quite useful, but the high complexity remains. The exponential time performance often remains even if one is ready to accept an approximate solution. This means that an important factor of exponential time is our desire to guarantee satisfactory results for all cases, including the worst one.

The reason is that the worst case can be very bad, if the family of functions to be optimized is large. Therefore, many applied global and discrete optimization problems are solved using heuristics. The heuristics are defined as some simple functions that roughly predict the consequences of decisions in the optimization process. We use randomized decision procedures if we wish to assure convergence to the global minimum, in the probabilistic sense, by a multiple repetition of randomized decisions.

If one knows or expects that some heuristics "work" well, then one may further enhance the efficiency of search by defining decision probabilities as a function of those heuristics. We may consider merely one specific heuristics or use a "mixture" of different heuristics.

We adapt the decision function to the given family of optimization problems by optimizing the randomization parameters through repeated optimization. We may do it for each problem or merely for one or several problems belonging to some "learning" set.

We fix a randomized decision function, except for some parameters that are defined as the randomization parameters. We repeat the decision procedure several times (for given values of the randomization parameters) and accept the best outcome as the result. We optimize the continuous randomization parameters to make the search more efficient. Thus we replace the original optimization problem by an auxiliary problem involving continuous stochastic optimization. We solve the auxiliary problem by means of Bayesian algorithms of global optimization, hence we assume some a priori distribution on a set of randomized heuristics ⁸.

The traditional way is to define the a priori distribution on a set of functions to be minimized. The definition of this distribution on the set of heuristic decision rules allows including of expert knowledge and accelerate the search. The reason is that we obtain here two ways of introducing expert knowledge.

⁸Using the Bayesian algorithms to optimize the parameters of randomized heuristics we tacitly assume an a priori distribution on a set of these parameters.

The first, is by defining an a priori distribution, while the second, a new one for the Bayesian Approach, is by selecting heuristics.

BHA may be applied to both global continuous optimization and discrete optimization. The difference is that in the continuous optimization all the "neighborhoods" and distances are uniquely defined. It is not so in the discrete case. This introduces some additional uncertainty while defining the a priori distribution directly on a set of functions of discrete variables. Using BHA for discrete optimization this uncertainty is avoided by defining the a priori distribution on a set of randomized heuristics instead of the set of objective functions. That makes the application of BHA to discrete optimization problems more attractive as compared to the traditional BA.

Many well known discrete optimization techniques belong to the MMA meaning that they are oriented towards the maximal deviation. Usually, these techniques are referred to as "exact methods". Each approach has advantages and limitations. The comparison by a single criterion is hardly possible, because both MMA and BA (including BHA) consider completely different criteria. This applies both to the theoretical and the experimental comparison.

For example, the investigation of asymptotic results is a natural part of the worst case analysis. In the Bayesian analysis the meaning of asymptotic performance is not so clear, the expected value after a finite number of observations is more important. It also changes the investigation methods. Asymptotic behavior is a domain of theoretical analysis: one cannot define asymptotic properties by computing. The estimation of expected values from limited data usually involves a lot of computing.

The main advantage of the Worst Case Analysis is that we obtain upper bounds on the deviation. The main disadvantage is the orientation towards the worst possible conditions. If the family of objective functions is large, then many iterations are needed to obtain sufficiently low upper bounds (see Table 12.7). That is the natural "cost" of performance guarantees.

The main advantage of the Bayesian Approach is the orientation towards the average conditions. An additional advantage of the Bayesian Heuristic Approach is the possibility of including expert knowledge in a natural and convenient way. The potential ability to "learn" is also a positive feature of the BHA. By learning we mean that randomization parameters, that are optimal for some problems of the given family, will be "good enough" for the rest of the family (see Section 3.4).

The main disadvantage is that we cannot define guaranteed bounds on the deviations. That is the "price" we have to pay for the advantages of the Bayesian Approach.

1.7.1 Illustrative Examples

Several well known examples are considered in this book to show the advantages and the disadvantages of the Bayesian Heuristic Approach. During the comparison we ignore the "learning" ability of Bayesian methods, meaning that we do not use the optimal randomization parameters of the previous problem as a "starting point" for the next problem.

The knapsack example is the most favorable one for deterministic and exact methods. Tables 12.1 and 12.2 describing the knapsack example suggest that the most efficient method seems to be close to the deterministic heuristics, if about 1% accuracy is required. Table 12.3 shows that the exact methods can be more economical, if the average accuracy is needed much higher than 1%. The knapsack example is a case when the Bayesian Heuristic Approach is not very useful.

The flow-shop example is known to be very challenging for exact methods. Table 12.7 shows that for a flow-shop example the average deviation of the standard MILP techniques exceeds that of the Bayesian heuristics many times for the comparable amount of calculations. The best heuristics happens to be the well known Gupta algorithm (12.11). The Bayesian improvement over pure Gupta heuristics is significant (keeping the amount of calculations about the same), (see Table 12.8). The flow-shop example shows the Bayesian Heuristic Approach in the most favorable light.

The travelling salesman example shows that if a heuristic is really good, then the Bayesian approach improves the pure heuristic (similar to [87]) only by 1 – 2% (see Table 12.5 and Table 12.6). Nevertheless, that might be important in some applications.

For both the knapsack problem [69] and the travelling salesman problem obeying "triangle inequality" [19], there exist polynomial time algorithms which provide fixed error bounds. For certain problems, no polynomial time approximation scheme can exist, unless there exists a polynomial time exact algorithm [50]. The flow-shop problem seems to belong to the family of problems difficult for approximation [54]. The difficulty of obtaining guarantee bounds is apparently

one of the factors explaining why the Bayesian heuristics work well in the flow-shop problem but not in the knapsack and travelling salesman problems.

This means the flow-shop is some type of "Bayesian" problem, while the knapsack and the travelling salesman problems are not. How to "recognize" the "Bayesian" problem is an important task of future investigation. By "Bayesian" we understand the problems where the Bayesian Heuristics Approach works better, as compared with the exact methods and pure heuristics.

The parameter grouping example shows that the Bayesian approach considerably improves the initial temperature of the well known Simulated Annealing techniques (see Figure 12.5).

We also discuss how to apply the Bayesian heuristics to the job-shop problem, and how to extend the Bayesian heuristic techniques to the stochastic case. The reason for this discussion is merely to show the potentials extending the Bayesian Heuristic Approach. No computing results are presented for these two problems.

The scheduling of batch processes [128] is considered in detail for different reasons. This example is very close to real life applications. In addition, the example shows that in some cases it is convenient to reduce the discrete optimization problem to the global optimization of continuous variables.

We approach this problem by including logical conditions and discrete parameters into some algorithmically defined objective and constraints. Then we apply the Bayesian Heuristic Approach using penalty functions. Table 14.1 and Figure 14.1 represent a simple illustrative example of batch scheduling.

1.8 DYNAMIC VISUALIZATION APPROACH: INTERACTIVE ANALYSIS

We describe some dynamic visualization techniques because DVA could be useful while solving the ill-defined optimization problems. We define an optimization problem as "ill-defined" if we update the objective and the model during the optimization process. It implies a necessity to define the objective and the

model interactively. Considering various real life optimization problems we see that many of them are "ill-defined".

Visualization is regarded as a decision optimization tool in problems where the model and/or the objectives are not well defined. We investigate four specific problems representing different degrees of determination.

The first problem concerns a smooth dynamic representation of data collected at fixed locations. In the example we want to minimize deviations from constant temperature in space and time.

The second and the third problems are a dynamic representation of observations in the form of averages over regions in space and time, and they are exemplified by epidemiological data. We are looking for spatial-temporal patterns that can suggest the most efficient ways for prevention and control.

The fourth problem may be referred to as visual indexing. We make an exploratory analysis of a large collection of complex objects. The example involves the application of a dynamic index to a collection of 30,000 images. We search for the "most interesting" subsets of images via visual inspection of the index.

In all the four cases we define appropriate techniques for the visual representation. We describe the software and hardware. The software and a videotape which displays the results may be acquired from Audris Mockus, e-mail: *audris@bell-labs.com*.

1.9 LIST OF APPLICATIONS

We apply BA for the following problems of continuous global optimization:

- modeling and yield maximization of electric circuits;
- optimization of shock-absorber;
- estimation of parameters of an immunological model;
- estimation of parameters of bilinear time series;
- estimation of parameters of the time series describing the exchange rates;
- search for the equilibrium in a competitive economic model;

- optimization of composite laminates;
- minimization of molecule potential energy;
- optimization of thermostable polymeric composition.

Some useful heuristics are described and potential abilities of BHA are discussed in the following network optimization problems:

- electric circuits;
- power grid;
- artificial neural networks.

We apply BHA to the following discrete optimization problems:

- knapsack;
- flow-shop;
- travelling salesman;
- parameter grouping.

We illustrate the dynamic visualization techniques by the following examples:

- smooth dynamic representation of data collected at fixed locations (with a view to minimize deviations from the constant temperature in space and time);
- dynamic representation of observations in the form of averages over regions in space and time, exemplified by epidemiological data (we are looking for spatial-temporal patterns that can suggest the most efficient ways of prevention and control);
- visual indexing, a dynamic index to a collection of 30,000 images (in search for the "most interesting" subsets of images by visual inspection of the index).

1.10 BATCH PROCESS SCHEDULING

Most of these examples are regarded as illustrations how to apply the techniques developed in this research. We describe one example separately and in detail. That is the scheduling of batch processes. One reason for this is that batch scheduling can be considered either as a continuous or as a discrete optimization problem. Since batch scheduling is an important and well known engineering problem, one may conveniently compare BHA with the results of other approaches.

A general framework for modeling and optimization of short-term scheduling problems in multi-product/multi-purpose batch chemical plants is described. We model time events in the schedule directly by a Non-Uniform Discrete-Time Model (NUDM) thus making the model clear and convenient for optimization purposes.

Batch processes can be represented by a state-task network. The problem is formulated as a Mixed Integer Non-Linear Program (MINLP) and the Bayesian Heuristic Approach is used to design some new global optimization techniques.

In Chapter 14, BHA is applied for the batch scheduling using the polynomial randomization (see Sub-section 11.5.1) of the well known Material Requirements Planning (MRP) heuristics.

In Chapter 15, BHA is applied to randomize the heuristics by means of simulated annealing techniques. In this chapter, we compare the results of Non-Uniform Discrete-Time Model with the traditional Uniform Discrete-Time Model. We consider several examples (including two well-known ones) and show the advantages and disadvantages of each approach.

In Chapter 16, we reduce the MINLP to a Mixed Integer Linear Programming (MILP) problem and consider how to solve it using BHA in Genetic Algorithms (GA).

1.11 GLOBAL OPTIMIZATION SOFTWARE

The global optimization software was developed considering the results of international "competition" of different algorithms of global optimization (see [30]). Some experience in real life optimization problems was also used in selecting the set of optimization algorithms (see the "application" chapters of this book). The set of algorithms of global optimization includes four versions of the Bayesian search, a version of clustering, a version of a uniform deterministic grid, and a version of pure Monte-Carlo search.

There are three local optimization methods. One method is of the variable metric type with Lagrange multipliers and penalty functions for constrained optimization of smooth functions (see [137]). The second method is of the simplex type of Nelder and Mead with penalty functions for constrained optimization of non-differentiable functions (see [61]). The third one is a stochastic approximation type with Bayesian step size control for noisy functions [100].

All global methods optimize in a rectangular region. Therefore we represent linear and nonlinear inequality constraints in some form of penalty functions. The same also applies to local methods of the stochastic approximation type.

In local methods of the simplex and variable metrics types, linear and nonlinear constraints can be defined directly. This may in fact be done by subroutines for constraints, supplied by the user.

The LINUX C++ version of the global optimization software GM is in the enclosed DOS format disk in the archive file 'gmc.tgz'. The archive 'gmc.tgz' includes illustrative examples of applications to knapsack and flow-shop problems, too.

In the files 'gmfl.tgz' and 'gmf.arj' are the portable Fortran 77 versions of GM for LINUX and DOS compilers, correspondingly.

In the directory 'classroom' there is a "class-room" version of GM for HP-UX. The "server" software is in 'classroom/server/gmhp.tgz', the "user" software is in 'classroom/user/Makefile'.

The illustrative examples of application of the software to optimization and predicting of exchange rate models are in the files 'arma.tgz' and 'arfima.tgz' (see README file in the disk). The software may also be acquired from Jonas Mockus, e-mail:

jonas@optimum.mii.lt

jonas.mockus@ktl.mii.lt.

2

INFORMATION-BASED COMPLEXITY (IBC) AND THE BAYESIAN HEURISTIC APPROACH

2.1 INTRODUCTION

Some concepts of Information Based Complexity are applied in global and discrete optimization, assuming that only partial information about the objective is available. We gather this partial information by observations and use the traditional IBC definitions and notions while defining formal aspects of the problem. The Bayesian framework is used to consider less formal aspects, like expert knowledge and heuristics, A parallel computing strategy is considered to overcome the computational difficulties in using the Bayesian Heuristic Approach.

The objective of this chapter is merely to discuss the relation between the concepts of IBC and the Bayesian Approach (see [75]) including BHA, while applying those concepts in global and discrete optimization. Specific techniques of BA and BHA are discussed in separate chapters of this book.

2.2 OUTLINE OF IBC

2.2.1 General Objective

We introduce some concepts of Information Based Complexity (IBC) [113, 149]. We follow the development given in [150] while using the IBC notation and definitions.

Let S be a mapping

$$S : F \rightarrow G \quad (2.1)$$

where F is a subset of a linear space and G is a normed linear space. The aim is to compute the approximation to $S(f)$ for all f from F .

2.2.2 Information Operations

Since, typically, f is an element from an infinite-dimensional space, it cannot be represented on a digital computer. We, therefore, assume that only partial information¹ about f is available. We gather this partial information about f by computing information operations $L(f)$, where $L \in \Lambda$ ². The class Λ denotes a collection of information operations that may be computed.

For each $f \in F$, we compute the number n of information operations from the class Λ . Let

$$N(f) = (L_1(f), L_2(f), \dots, L_n(f)) \quad L_i \in \Lambda \quad (2.2)$$

be computed information about f where L_i can be chosen adaptively. That is, the choice of L_i may depend on the already computed $L_1(f), L_2(f), \dots, L_{i-1}(f)$. We do not consider termination criteria, thus we fix the number n of information operations.

$N(f)$ is referred to as the information about f , and N is an information operator. In general, N is many-to-one, and that is why it is impossible to recover the element f , knowing $y = N(f)$ for $f \in F$. For this reason, the information N is defined as partial.

2.2.3 Algorithm of Approximation

Having computed $N(f)$, we approximate $S(f)$ by an element $U(f) = \phi(N(f))$, where $\phi : N(F) \rightarrow G$. The mapping ϕ is referred to as an algorithm.

The definition of the error of the approximation U depends on the setting. In the worst case setting

$$e(U) = \sup_{f \in F} \|S(f) - U(f)\|. \quad (2.3)$$

¹For simplicity, we do not consider contaminated information (noisy observations).

²Later we refer to these operations as observations.

while in the average case setting, given a probability measure μ on F , the error is defined as follows:

$$\begin{aligned} e(U) &= \mathbf{E}_{\mathbf{F}} \|S(f) - U(f)\| \\ &= \int_{f \in F} \|S(f) - U(f)\| \mu(df). \end{aligned} \quad (2.4)$$

2.2.4 Cost of Computing

Suppose that for each $L \in \Lambda$ and for each $f \in F$, the computation of $L(f)$ costs a unit of computing. We define that unit as the observation cost. Let $cost(N, f)$ denote the cost of computing the information $N(f)$. Note that $cost(N, f) \geq n$, and the strict inequality may occur, since adaptive selection of L_i may require some operations in addition to the cost of n observations.

Knowing $y = N(f)$, we compute $U(f) = \phi(y)$ by combining the information $L_i(f)$. Let $cost(\phi, y)$ denote the cost of the operations needed to compute $\phi(y)$.

The cost of computing $U(f)$, $cost(U, f)$, is given by

$$cost(U, f) = cost(N, f) + cost(\phi, N(f)). \quad (2.5)$$

In the average case setting

$$cost(U) = \int_F cost(U, f) \mu(df). \quad (2.6)$$

In subsequent development we shall divide $cost(U, f)$ into two parts: the cost of observations n and the cost of auxiliary computations a needed for adaptive selection of L_i and for computation of the algorithm $\phi(y)$. Usually we are certain how to define the observation cost but are not certain about the cost of auxiliary computations. Therefore, we shall consider the cost of auxiliary computations only in qualitative terms, such as "low", "high", etc.

In the IBC framework, the ϵ -complexity is defined as the minimal cost among all U with an error of at most ϵ

$$comp(\epsilon) = \inf\{cost(U) : U \text{ such that } e(U) \leq \epsilon\}. \quad (2.7)$$

2.2.5 Exponential-Time Algorithms

We define an algorithm as exponential-time, if the solution time

$$T_s \geq C2^s, \quad (2.8)$$

where $C > 0$ and the parameter s defines the "complexity" of a problem.

In discrete optimization problems, the complexity s_n is understood as the number of variables n , meaning that $s_n = n$. Then, from expression (2.8)

$$T_n \geq C2^n. \quad (2.9)$$

In problems of real function optimization, we understand the complexity s_n as the index n of guaranteed accuracy of approximation a_n , therefore $s_n = n$. We define the accuracy as the error limit³ ϵ_n , where the approximation a_n and the accuracy ϵ_n are related by the expression,

$$|a_n - \min_x f(x)| \leq \epsilon_n. \quad (2.10)$$

Here the error limit $\epsilon_n = 2^{-n}$ (see [79]).

An algorithm is of exponential-time, if to obtain the accuracy ϵ_n of approximation a_n one needs exponential time of computation. From this definition and expressions (2.8),(2.10) it follows that

$$T_n \geq C2^n. \quad (2.11)$$

Thus we arrived at seemingly identical definitions of exponential-time algorithms for both the discrete and the real case (see expressions (2.9) and (2.11)). The difference lies in the different definitions of complexity s_n .

We call an algorithm to be of polynomial time, if there exist $C > 0$ and m such that the solution time

$$T_s \leq Cs^m. \quad (2.12)$$

Here s defines the complexity of the problem. These notions are related to the well known *NP*-complete class. If a problem of this class needs polynomial solution time, then all the others need that, too⁴. If one problem of the *NP*-complete family can be solved in polynomial time, then all the others can, too. No polynomial time algorithm has been found for any *NP*-complete problem.

³We assume that small error limit implies high accuracy.

⁴A rigorous definition of the *NP*-complete class in both the continuous and discrete case is in [79].

2.2.6 Radius of Information

The traditional IBC theory makes a distinction between information and algorithm. As we have already explained the approximation $U(f)$ is computed by combining information operators from the class Λ . Let $y = N(f)$ denote the computed information. The set $S(N^{-1}(y))$ consists of all elements from G which are indistinguishable from $S(f)$. Since $U(f)$ is the same for any f from the set $N^{-1}(y)$, the element $U(f)$ must serve as an approximation to any element g from the set $S(N^{-1}(y))$. The quality of approximation $U(f)$ depends on the "size" of the set $S(N^{-1}(y))$. Therefore IBC theory defines the radius of information $rad_{Info}(N)$ while considering the set $S(N^{-1}(y))$ for $y \in N(F)$. In the worst case setting that means the maximal radius is given by:

$$rad_{Info}(N) = \sup_{y \in N(F)} rad(S(N^{-1}(y))). \quad (2.13)$$

Here $rad(A) = \inf_{x \in G} \sup_{a \in A} \|x - a\|$.

In the average case setting, the radius of information should be defined as the average radius. However, a formal definition of the average radius is more complicated than appropriate for the present setting (see [113]). We shall not use the information radius while considering optimization problems. We merely introduce the information radius as an elegant concept of IBC.

2.2.7 Bayesian Risk

We shall minimize the Bayesian risk $R(\mu)$ defined by the condition

$$R(\mu) = \inf_U e(U) \quad (2.14)$$

Note that infimum is taken over all possible U , where U is identified with a pair (N, ϕ) , where N is the information and ϕ is an algorithm that uses this information. This means that the infimum is taken over all the information N consisting of information operations from the class Λ , and over all algorithms ϕ that use N . We next apply these concepts to global optimization.

2.3 IBC AND GLOBAL OPTIMIZATION

2.3.1 Observations

Let F be a family of continuous real functions f defined on the m - dimensional hypercube $A = [-1, 1]^m \subset R^m$, thus $G = R^m$. Let

$$S(f) = x^*(f) = x^* = \arg \min_{x \in A} f(x). \quad (2.15)$$

The information is obtained by the information operator

$$N(f) = (f(x_1), f(x_2), \dots, f(x_n)). \quad (2.16)$$

Here the points x_i are adaptively chosen and the number n is given. A pair $z_i = (y_i, x_i)$, where $y_i = f(x_i)$, is defined as an observation. A vector $z(n) = (z_1, z_2, \dots, z_n)$ is defined as an observation vector.

Denote by $x_{n+1} = x_{n+1}(\phi)$ the final decision by the algorithm ϕ using the observation vector $z(n)$. The result of the final decision is denoted as $y_{n+1} = f(x_{n+1})$ and the result of the last observation is denoted by x_n . The index $i = 1, \dots, n$ denotes current observation.

Since adaptive information is considered, the coordinates of the next observation x_{i+1} depend on the already observed results $z(i) = (z_1, z_2, \dots, z_i)$. The observation vector $l(N, f) = z(i)$ is the information about f obtained after i observations. In general, $z(i)$ is multi-valued, and that is why it is impossible to recover the element f , knowing $z(n) = N(f)$ for $f \in F$.

2.3.2 Decision Algorithms

Having computed $z(n) = N(f)$, we approximate $S(f)$ by an element $x_{n+1} = U(f) = \phi(N(f))$, where $\phi: z(n) \rightarrow A$. We define x_{n+1} as the final decision.

If we wish to perform each observation in an optimal way we have to define the algorithms of sequential decisions, too. A mapping $\phi_i^n: z(i) \rightarrow A$ is defined as an algorithm of sequential decision, if $i < n$. Thus we define each observation $x_i = \phi_{i-1}^n(z(i-1))$, $i = 1, \dots, n$ ⁵.

⁵The notion of sequential decisions is indirectly included while defining the adaptive information operations L_i in the traditional IBC framework [113].

2.3.3 Errors

The error of approximation U of the global minimum in the average case setting, given a probability measure μ on F ,

$$e(U) = \int_F w(S(f), U(f)) \mu(df). \quad (2.17)$$

Assume a linear loss function w to be:

$$w(S(f), U(f)) = f(U(f)) - f(S(f)). \quad (2.18)$$

Here $f(S(f)) = f(x^*)$ is the global minimum of f , and $f(U(f)) = f(x_{n+1})$ is its approximation after n observations.

The second component $f(S(f))$ of expression (2.18) does not depend on the decision algorithms ϕ and ϕ_i^n . Therefore we truncate expression (2.18) by omitting the second component

$$w(U(f)) = f(U(f)) = f(x_{n+1}). \quad (2.19)$$

Then from expressions (2.17) and (2.19)

$$e(U) = \int_{f \in F} f(U(f)) \mu(df). \quad (2.20)$$

Note that the non-negative linear loss function (2.18) is a specific feature of optimization problems. Therefore the average error is defined by expression (2.20) omitting the constant term $\int_{f \in F} f(S(f)) \mu(df)$ where $f(S(f)) = f(x^*)$. That is convenient considering the theoretical cases because the expression of average of the minimum $f(x^*)$ is much more complicated as compared with that of average of $f(x_{n+1})$ at a fixed point x_{n+1} ⁶. This also justifies the direct comparison of different approximate algorithms using "truncated" expression (2.20) instead of traditional expression (2.17). That is important while comparing various approximate methods by solving *NP*-complete problems where exact algorithms need exponential time.

⁶The reason is that the optimal x^* depends on both the objective f and the algorithm ϕ .

2.4 BAYESIAN APPROACH

2.4.1 Sequential Decisions

After the observation i we choose the next $x_{i+1} \in A$ minimizing the expected error $R_i(x)$:

$$x_{i+1} = \arg \min_{x \in A} R_i(x),$$

$$R_i(x) = \int_{N^{-1}(z(i))} f(U(f)) \mu(df|z(i)). \quad (2.21)$$

It is supposed that the error is a continuous function of x and that the set A is compact. Condition (2.21) defines the sequential decision rule $\phi_i^n : z(i) \rightarrow A$.

Using the statistical language, one may say that the measure $\mu(f|z(i))$ is an a posteriori measure that represents our belief in the distribution of elements $f \in N^{-1}(z(i))$ which are indistinguishable after the information $z(i) = N_i(f)$ has been obtained. Here $N_i(f) = (f(x_1, \dots, f(x_i)))$ denote information operator (2.16) truncated at the observation $i \leq n$.

Clearly, defining algorithm (2.21) one has to solve kind of a multi-dimensional nonlinear dynamic programming problem. The reason is that making a decision at some stage $i < n$ one has to predict what information will be obtained later. This means that we should predict all the pairs $z_j = (x_j, y_j)$, $i < j < n+1$. We cannot expect to obtain the complete multi-stage solution. Thus we consider various ways of simplifying the problem.

2.4.2 Simplified Decisions

Assuming that the current observation is the last one ($i = n$) we may reduce the sequential decision rule (2.21) to a sequence of non-sequential problems⁷. We define a non-sequential decision, based on the assumption that $i = n$, as "zero-step" approximation [102].

In doing so one may oversimplify. If the minimal conditional expectation happens to be at some observation point, then the optimization stops. An example is the Wiener process. Consider a family of continuous functions equipped with

⁷Corresponding to the optimal error algorithm ϕ defined by [113].

the Wiener measure μ [168]. Usually

$$\arg \min_{x \in A} \mu_i(x) = x_{j(i)}, \quad (2.22)$$

where $\mu_i(x)$ is the conditional expectation of $f(x)$ given $z(i)$, and

$$x_{j(i)} = \arg \min_{1 \leq j \leq n} f(x_j). \quad (2.23)$$

It follows from expressions (2.21) and (2.22) that

$$x_{i+1} = x_{j(i)}. \quad (2.24)$$

This means that the optimization stops (see condition (2.22)), and one may be far away from any minimum. Therefore we consider a "one-step" approximation, later on assuming that the next observation is the last one: $i = n - 1$. This means that we use sequential decisions, but we look only one step ahead.

Theoretically one may look two steps ahead, three steps ahead, and so on. However, we do not see any specific advantages to look more than one step ahead. Besides, we do not know any practical means to do that.

2.5 A POSTERIORI MEASURE

2.5.1 Traditional Case

If $N(f)$ is a measurable mapping and if F is a measurable subset of a separable Banach space [113], then one may define the a posteriori measure $\mu|z(i)$ by an a priori measure μ . Thus Kolmogorov's consistency conditions are retained and we obtain a powerful analytical tool. For example, one may show how the sample behavior depends on the a priori measure μ , etc. The conditions on μ providing the continuity, differentiability, and Lipschitzian properties of sample functions are well known [23].

For example, the sample functions $f(x)$ of some stochastic process ξ are continuous (*mod P*)⁸ on $[0, 1]$ if the difference

$$\rho_h(t) = \rho(t+h, t+h) - \rho(t+h, t) - \rho(t, t+h) + \rho(t, t) \quad (2.25)$$

⁸The term (*mod P*) is short for "with probability $P = 1$ " and is convenient considering several different probability measures.

satisfies the condition

$$\rho_h(t) < \frac{K|h|}{\|\log |h|\|^q}, \quad (2.26)$$

where $q > 3$, $K > 0$, and P is a probability measure defining the stochastic process ξ . These results are important while using the proper sequential decisions as defined by equations (2.21).

2.5.2 Replacing Consistency Conditions

It is less clear why to keep the consistency conditions when one-step approximation is applied. Using one-step approximation means that the statistical model is changed after each observation. The consistency conditions imply updating information while keeping the same statistical model (considering the same stochastic function). Consequently the traditional consistency conditions seem almost irrelevant using one-step approximation. Therefore we replace them by some weaker conditions, namely, the continuity of Bayesian risk and simplicity of expressions (see [100]).

2.6 BAYESIAN HEURISTIC APPROACH

2.6.1 Why Heuristics?

An important advantage of the Bayesian approach is that one may include some expert knowledge. That is the main reason why we use the Bayesian approach. Applying the traditional Bayesian setup one may include the expert knowledge while defining an a priori measure μ . Looking at many real life decision techniques one may notice that the expert knowledge is usually exploited via some expert decision rules so-called heuristics.

There are hundreds of well known heuristic optimization techniques in engineering design and many other fields [115], [59]. Therefore we may extend the application of Bayesian approach if we find the ways how to include heuristic optimization into the Bayesian framework. Further we will consider one such way.

2.6.2 Defining Heuristics

Using BHA denote the original objective not by f but by v :

$$v = v(d), \quad d \in A \subset R^m. \quad (2.27)$$

Denote the original optimization problem by

$$d^* = \min_{d \in D} v(d). \quad (2.28)$$

A different notation of the objective helps to recognize what approach is used: the objective v means BHA, and the objective f means BA, or some other traditional approach not involving any heuristics.

Denote some additional objective as

$$h = h(d, v) \in H, \quad d \in D = D(v) \subset A. \quad (2.29)$$

Define function (2.29) as a heuristics. Define the following mapping as a randomized heuristic decision:

$$\delta_n^\pi : H \rightarrow D. \quad (2.30)$$

Mapping (2.31) defines the variable $d_n \in D$ as a function of heuristics h depending on some randomization measure π and the stage n :

$$d_n = \delta_n^\pi(h). \quad (2.31)$$

We refer to a sequence of heuristic decisions d_n as π -consistent if it converges to the global minimum x^* :

$$\lim_{n \rightarrow \infty} d_n = x^* \quad (\text{mod } \pi). \quad (2.32)$$

We consider randomized heuristic decisions because randomization is an easy way to obtain a convergence. Condition (2.32) holds, if the probability π to hit any ϵ -neighborhood in the continuous case (or any point in the discrete case) is positive (see Theorem 3.2.1).

Usually we cannot specify exactly what heuristics h and what randomization π would be preferable. In such cases, it is natural to consider a "mixture" of heuristics h_s and/or a mixture of randomizations π_l . However, we are not certain about the best "weights" of the mixture. The "weight" of a mixture component implies a probability of using this component. It is a some type of a lottery where one may "win" some particular heuristics or/and some specific

randomization procedure. In this lottery weights define the probabilities to "win".

We represent heuristics by a vector $h = (h_s, s = 1, \dots, S)$ and randomization measures by a vector $\pi = (\pi_l, l = 1, \dots, L)$. Denote weights of the components h_s and π_l by a vector $x \in X$:

$$x = (x_i, i = 1, \dots, n), \quad \sum_i x_i = 1, \quad x_i \geq 0, \quad n = S + L. \quad (2.33)$$

A convenient way of representing the uncertainty about the mixture weights is by an a priori measure μ_H on X .

Applying the Bayesian Heuristic Approach (BHA) the solution of the original optimization problem (2.28) may be divided into two parts

- In the first part minimizing the auxiliary objective $f(x)$ one obtain the optimal mixture of randomized heuristics

$$x^* = \arg \min_{x \in X} f(x) \quad (2.34)$$

given the a priori measure μ_H of weights $x \in X$.

- In the second part we the original objective (2.27) is optimized using randomized heuristic decisions (2.31) defined by this mixture.

Here the auxiliary objective $f(x)$ defines the best results one obtains by multiple repetition of randomized decisions given the weights x .

2.6.3 Optimizing Heuristics

One may regard this two-part procedure as a reduction of the original problem of objective $v(d)$ optimization to an auxiliary problem of mixture weights x optimization. By mixture weights we define the probabilities to use different heuristics or/and different randomizations. The best results of repeated Monte Carlo simulation applying some consistent heuristics h to a function f using the weights x are defined by $f(x)$. These techniques are referred to as a Bayesian Heuristic Approach (BAH) [94]. One may see that the number of weight vector components is defined only by the number of different heuristics and different randomization procedures.

Thus the dimensionality m of the original problem $\min_d v(d)$, $d \in R^m$ is not directly related to the dimensionality n of the corresponding BHA problem $\min_x f(x)$, $x \in A \subset R^n$, where $n < m$, as usual.

2.6.4 Discrete Optimization

There are some additional difficulties in defining an a priori measure directly on a set of the original discrete optimization problem. The reason is that the neighborhood relations are not uniquely defined on discrete sets. Those relations are important in defining the a priori measure. It is natural to assume the statistical relation weak between the distant points and strong between the close ones. We may apply this assumption only if we know the neighbors. The problem is that the a priori measure in discrete cases depends on the neighborhood definition what is not always clear.

We avoid this problem using the Bayesian Heuristic Approach. Here the a priori measure is defined on a continuous set of weights of randomized heuristic decisions. This is an additional advantage of applying BHA to an discrete optimization problems.

2.6.5 "Learning" Heuristics

The efficiency of Bayesian Heuristic approach is enhanced by "learning" techniques, common in pattern recognition, neural networks, etc. By "learning" we mean the optimization of x using a "learning set" (some selected members of the objective function v of a family V). We apply these parameters to the other members of the family V later on (with or without additional optimization) (see [82]). That is important in designing "on-line" techniques. The learning effect is investigated in the sequel while considering the flow-shop problem (see Section 12.3).

2.6.6 Comparison with Traditional Approaches

In traditional approaches (both IBC and BA) the a priori measure μ is defined on a set of original objective functions (see expression (2.20)). The optimal decisions are determined by expressions (2.21).

Using BHA, we consider a different framework: we define the a priori measure μ_H on a set of randomized heuristic decisions directly. This means that we skip the most difficult part of the problem. The reason is that applying BA to optimize the original problem directly, we:

- consider equations (2.21) of the same high dimensionality as that of the original problem;
- include the expert knowledge while defining an a priori measure on a set of functions to be optimized.

Applying BHA to solve the original problem indirectly by optimizing a "mixture" of heuristics, we:

- consider equations (2.21) of lower dimensionality that is equal to the number of different heuristics and randomization techniques which usually is not too large;
- include the expert opinion in a natural and efficient way while defining heuristics ⁹.

Now let us discuss in short how to "restore" the traditional BA problem, given the BHA problem. Theoretically one may describe a BHA problem by the triplet (μ_H, π, h) , where μ_H is an a priori measure defined on a set X of mixtures x (see (2.33)), π is a randomization measure, and h is a heuristic. One may describe a BA problem merely by an a priori measure μ .

It is possible to "restore" the BA problem in some simple cases. By restoring we understand a definition of an a priori measure μ on a set of objective functions such that the optimal solution of two parts of BHA problem (2.31) and (2.34) satisfies optimality conditions (2.21) of the BA problem. In other words, restoring the BA problem we are looking for a single measure μ such that yields BA decisions the same as the BHA triplet (μ_H, π, h) .

Consider a trivial example of "irrelevant" heuristic h . We define a heuristic as irrelevant if there exists no relation between h and the original objective v . In this case, under some obvious conditions, the randomized heuristic decision

⁹In addition to that, we also include the expert knowledge by defining an a priori measure on a set of "mixtures" of heuristics. This means that BHA provides two ways to include the expert knowledge. Traditional BA provides only one way

(2.31) of BHA corresponds to simple Monte-Carlo search. It is easy to see that Monte Carlo search satisfies optimality condition (2.21) of BA, too, if the a priori measure μ is of "white noise" type. This means that "white noise" decisions in the traditional BA framework correspond to the decisions of irrelevant heuristics using the BHA. The possibility of restoring theoretically BA problem from BHA problem in non-trivial cases is doubtful. However, there is no apparent reason for such restoration.

Note that a related "restoration" was considered experimentally while investigating decisions of experts solving a multi-modal problem (see [1]). It was shown that the heuristic expert decisions (the BHA problem) roughly correspond to the Bayesian ones assuming some a priori distribution (the "restored" BA problem).

2.7 PARALLEL BAYESIAN ALGORITHMS

2.7.1 Outline

Bayesian algorithms perform three types of computations:

- observation: here we define the objective $f(x)$ given $x = x_n \in A$;
- forecasting: here we optimize the Bayesian risk $R(x)$, thus defining the next observation point x_{n+1} ;
- heuristics evaluation: here the best results of a randomized heuristics by Monte Carlo simulation are defined.

One may apply parallel algorithms in all the three cases.

2.7.2 Parallel Risk Optimization

Usually one optimizes the risk $R(x)$ defined by expression (2.21) ¹⁰ generating M points x by some simple covering techniques (such as Monte Carlo or LP-sequences). If there are K processors, then one may calculate $R(x)$ at K

¹⁰Some simplified expressions (see condition (4.11)) are used, as usual.

different points x at the same time. Obviously $M \geq K$ and M should be divisible by K . The result of risk optimization is a set $I_R(K)$ of K best points: $l \in I_R(K)$ if $R(x_l) \leq R(x_j)$, $j \notin I_R(K)$. That is one iteration. We repeat the same procedure at each iteration.

2.7.3 Parallel Observations

At each iteration we observe a set $I_R(K)$ of K best points. Each observation is performed in parallel by different processors. We use all available observations defining risk $R(x)$ in the next iteration and continue the optimization until we reach the limit observation number N which has to be divisible by K . The processor number K is assumed much less as compared with the total observation number N . Otherwise, we should consider different techniques.

2.7.4 Parallel Heuristics

Denote by $f_K(x)$ the best results we obtain repeating K times the "mixture" of randomized heuristics defined by the parameter vector x [94, 95]. For example, x_0 may denote a "weight" of the uniform component of randomization, x_1 may denote a weight of the linear component, and x_2 may denote a weight of the quadratic component. If there are K processors, then one may perform all the K "repetitions" at the same time. If more repetitions are needed, then one may choose any repetition number divisible by K .

2.8 CONCLUSIONS

It is convenient to apply the traditional IBC definitions and notions directly while formulating the global optimization problem in general. If we wish to include less formal aspects, such as expert knowledge, then the Bayesian framework seems more natural. The Bayesian techniques are fit for the optimization of stochastic problems and for parallel computing.

3

MATHEMATICAL JUSTIFICATION OF THE BAYESIAN HEURISTICS APPROACH

3.1 INTRODUCTION

The main objective of BHA is to increase the efficiency of heuristics by including them into the Bayesian framework. A natural criterion of algorithm efficiency is the results of real life problems [30, 44, 115, 63]. Needless to say, one should use any available mathematical justification, too. An important condition is to regard the theoretical and experimental results in right proportions.

For example, it would be wrong to assume that an algorithm that converges to the optimum is always superior to the one that does not. However, it would be equally wrong to ignore the convergence. A realistic approach is to consider the convergence as one of important criteria. The convergence should be regarded merely as a necessary condition for an algorithm to pretend to some mathematical justification. As usual, it is not difficult to prove some convergence, under appropriate assumptions.

A disadvantage of convergence as a criterion is that it is too "easy" for a meaningful comparison. Many methods converge, including obviously inefficient, such as Monte-Carlo search. Therefore we regard the convergence as a first stage of theoretical justification.

Some inequalities could be important as a second stage of theoretical comparison and justification of global and discrete optimization techniques. It is shown on simple examples that we obtain better results by randomizing heuristics and by "mixing" different randomizations.

A third stage of theoretical justification is to show some useful properties, typical of the given algorithm. For example, in a Bayesian case a specific property is minimization of the average deviation (by definition). Another important feature is the "learning" ability of BHA.

3.2 DEFINITION OF CONVERGENCE

3.2.1 Convergence of Deterministic Techniques

Using the notation of the previous chapter we consider the information vector

$$N(f) = (f(x_1), f(x_2), \dots, f(x_n)) \quad (3.1)$$

with the points $x_i \in A \subset R^m$ adaptively chosen and the number n given.

We define a pair $z_i = (y_i, x_i)$, where $y_i = f(x_i)$ as an observation. We refer to a vector $z(n) = (z_1, z_2, \dots, z_n)$ as an observation vector.

The traditional convergence definition is as follows. An algorithm (3.1) converges to the optimum x^* if for any objective $f \in F$ and any positive ϵ there exists n_ϵ such that

$$\nu_n = \nu_n(f, d) \leq \epsilon, \text{ if } n \geq n_\epsilon. \quad (3.2)$$

Here the deviation $\nu_n(f, d)$ defines the difference between the final decision $x_{n+1}(d)$ ¹ using the search algorithm d while optimizing the objective function f .

Condition (3.2) implies that

$$\sup_{f \in F} \nu_n(f, d) \leq \epsilon, \text{ if } n \geq n_\epsilon. \quad (3.3)$$

Therefore, the convergence condition (3.2) belongs to the worst case analysis.

Convergence definition (3.2) is not unique, it depends on the definition of the deviation $\nu_n(f, d)$. In this book we define a deviation as the difference between

¹Note that the index $n + 1$ and a reference to algorithm d defines the final decision made by algorithm d after n observations. A point of the last observation is denoted as x_n .

the objective values

$$\nu_n = \nu_n(f, d) = f(x_{n+1}(d)) - f(x^*) \quad (3.4)$$

assuming that the final decision $x_{n+1}(d)$ of the decision procedure d is the best observation

$$x_{n+1}(d) = \arg \min_{1 \leq i \leq n} f(x_i). \quad (3.5)$$

Denote the result of final decision by $y_{n+1} = f(x_{n+1}(d))$.

The convergence often is determined assuming tacitly that the final decision is the last observation

$$x_{n+1}(d) = x_n, \quad (3.6)$$

and defining a deviation as the difference

$$\nu_n = \|x_n - x^*\|. \quad (3.7)$$

This definition is usual in local optimization. Different deviations define different convergences. A definition of convergence (3.2) using traditional conditions (3.6) and (3.7) implies that we reach the ϵ -vicinity of global minimum and will "stay" there. Therefore we refer to it as a "stay-convergence".

Convergence definition (3.2) using expressions (3.4) and (3.5) means that we "hit" the ϵ -vicinity of global minimum at least once. That is called a "hit-convergence". This definition is weaker since condition (3.5) defines a subsequence of $x_n(d)$.

We may regard the stay-convergence as a special case of the "convergence rate" (see expression (3.9)) if $\lim \Gamma = \infty$. If $\lim \Gamma = C < \infty$, then "convergence rate" condition (3.9) is weaker than that of stay-convergence (3.7) and (3.6).

Speaking of the convergence in this book we usually mean hit-convergence defined by expressions (3.2) and (3.4) assuming that the final decision is the best one (see condition (3.5)). Other convergence definitions are mentioned merely for comparison.

An algorithm (3.1) is said to converge super-linearly, if for any objective $f(x) \in F$ and any positive ϵ there exists n_ϵ such that

$$\frac{\|x_n - x^*\|}{\|x_{n-1} - x^*\|} \leq \epsilon, \text{ if } n \geq n_\epsilon. \quad (3.8)$$

Convergence definition (3.2) using conditions (3.6), (3.7) is natural and convenient in a local optimization, but not in the global one. The reason is that two very different points $\|x_1 - x_2\| \geq M$ may be equal $f(x_1) = f(x_2)$ (or almost equal) in the sense of the objective function, what is confusing. We avoid the contradiction by using conditions (3.4) and (3.5) in the convergence definition (3.2).

Following [100] we define the global optimization convergence rate as a limit of the relation of observation density γ_n^* around the global minimum to the average density γ_n . We denote this relation as $\Gamma_n = \gamma_n^*/\gamma_n$. Then, the "convergence rate"

$$\Gamma = \lim_{n \rightarrow \infty} \Gamma_n. \quad (3.9)$$

where Γ is determined by condition (4.12).

A different definition of the convergence rate of global random search algorithms is given in [126]. This shows that the average deviation is equal to $O(n^{-2/m})$. However, this result is applicable only while considering the neighborhood of x^* and assuming that the Hessian of $f(x^*)$ is positive definite. In contrast, one may apply the convergence rate definition (3.9) while considering all the area A .

3.2.2 Convergence of Randomized Techniques

Algorithm (3.1) is said to be random, if the observations depend on some probability measure P . A random algorithm converges (*mod* P) if for any objective f , any positive ϵ and δ , and all $m = 1, 2, \dots$ there exists $n_{\epsilon, \delta}$ such that

$$P(\sup_l \nu_{n+l} \geq \delta) \leq \epsilon, \quad n \geq n_{\epsilon, \delta}. \quad (3.10)$$

Random algorithm (3.1) is said to converge in probability P , if for any objective $f(x)$ and any positive ϵ and δ , there exists $n_{\epsilon, \delta}$ such that

$$P(\nu_n \geq \delta) \leq \epsilon, \quad n \geq n_{\epsilon, \delta}. \quad (3.11)$$

Here ν_n denotes the deviation defined by expressions (3.4)(3.5) or by alternative expressions (3.7) (3.6). Conditions (3.7) (3.6) are applied while considering the convergence of simulated annealing (see [154]) and stochastic approximation (see [39]) methods. We define the convergence using weaker conditions (3.5) (3.4) assuming that the final decision is the best one.

We define a gap radius ρ_n as a maximal distance to the nearest observation

$$\rho_n = \max_{x \in A} \min_{1 \leq i \leq n} \|x - x_i\|. \quad (3.12)$$

Considering optimization of a continuous objective $f(x)$, $x \in A$ on a compact set A using conditions (3.4)–(3.5) we replace the deviation ν_n by the gap radius ρ_n in convergence definitions (3.10) and (3.11)².

Then, from conditions (3.10) and (3.11) it follows

$$P(\sup_l \rho_{n+l} \geq \delta) \leq \epsilon, \quad n \geq n_{\epsilon, \delta} \quad (3.13)$$

and

$$P(\rho_n \geq \delta) \leq \epsilon, \quad n \geq n_{\epsilon, \delta}. \quad (3.14)$$

The gap radius (3.12) is a non-increasing function of m . This means, that

$$\sup_m \rho_{n+m} \leq \rho_n. \quad (3.15)$$

In this case stronger condition (3.13) is satisfied, too, if weaker condition (3.14) holds. Later on we shall consider mainly the randomization techniques that satisfy condition (3.14).

Gap-Avoiding Randomization (GAR)

Denote by $p_i(\alpha)$ the probability that an observation i will "hit" the region $\alpha \in A$. Denote by

$$r_i(\alpha) = 1 - p_i(\alpha) \quad (3.16)$$

a "no-hit" probability.

Denote by $r_{ij}(\alpha)$ a joint no-hit probability, meaning that both the observations i and j do not hit α . We refer to a randomization as GAR, if

$$r_{ij}(\alpha) \leq r_i(\alpha)r_j(\alpha). \quad (3.17)$$

²We do the same replacement for discrete optimization problems, too.

Theorem 3.2.1 *A Gap-Avoiding Randomization (GAR) converges, in the sense of condition (3.10), for any continuous objective $f(x), x \in A$ on a compact set $A \in R^m$ (or for an objective f defined on a discrete set A), if the condition*

$$p_i(\alpha) \geq 1 - \exp(-c/i) \quad (3.18)$$

holds for some $c > 0$.

Proof

From condition (3.17) the probability $r(n, \alpha)$ that not a single of n observations will hit the region α

$$r(n, \alpha) \leq \prod_{i=1}^n r_i(\alpha). \quad (3.19)$$

Denote

$$r(\alpha) = \lim_{n \rightarrow \infty} r(n, \alpha). \quad (3.20)$$

It follows from conditions (3.19) and (3.18) that

$$r(\alpha) = 0 \quad (3.21)$$

because

$$r(\alpha) = \lim_{n \rightarrow \infty} \exp(-c \sum_{i=1}^n 1/i) = 0. \quad (3.22)$$

This means that the gap radius ρ_n defined by expression (3.12) satisfies convergence condition (3.14) and consequently condition (3.13). Condition (3.13) implies the convergence (*mod P*) defined by condition (3.10) in the case of optimization of the continuous functions f on a compact set A .

That proves a "continuous" part of the theorem.

Using expression (3.4) we may formally replace the deviation ν_n by the gap radius ρ_n in convergence conditions (3.10) and (3.11) for discrete optimization problems, too. Therefore, the theorem conditions also hold, if the set A is discrete.

□

It is well known [151] that in the continuous case the convergence of random search algorithms follows from the assumption that $p(\alpha) > 0$ for each α of

a positive Lebesgue measure, where $p(\alpha) = 1 - r(\alpha)$. Theorem 3.2.1 is more specific, in the sense that it defines the sufficient conditions when the assumption $p(\alpha) > 0$ holds. Testing of these conditions is simpler, sometimes.

Convergence condition (3.18) is satisfied under the usual convergence conditions of simulated annealing [154]. The advantage of Theorem 3.2.1 is that it holds for any "initial temperatures". However this theorem ensures merely the hit-convergence (see conditions (3.10) (3.5) (3.4)) assuming that the final decision is the best one. The traditional simulated annealing convergence theorems [154] regard the stronger stay-convergence (see conditions (3.10) (3.7) (3.6)).

3.3 ADVANTAGES OF RANDOMIZED HEURISTICS

3.3.1 Optimizing a "Mixture" of Pure Heuristics and Monte-Carlo Search

Consider a given sequence of Monte-Carlo and pure heuristic search. For example, the symbol $(r, \dots, r, h, \dots, h)$ means that the Monte-Carlo search (denoted by r) is repeated several times and then multiple pure heuristics (denoted by h) is applied. Denote by $v_k(r, \dots, r, h, \dots, h)$ the average results in this case, where k is the number of repetitions including both Monte-Carlo and pure heuristics. Assume, for illustration, that

- the values of the objective function are uniformly distributed in a unit interval;
- results of pure heuristics may be expressed as

$$v_k(h, h, \dots, h) = y^0 \left(\delta + \frac{1 - \delta}{(k + 1)^2} \right) = \delta + \frac{1 - \delta}{(k + 1)^2} \quad (3.23)$$

regarding the starting point $y^0 = 1$.

The parameter $\delta \geq 0$ denotes an asymptotic residual value as $k \rightarrow \infty$. Thus, expression (3.23) roughly represents the results of some local search algorithm starting from the initial value y_0 and approximating a local minimum by δ .

After simple calculations we obtain the expression of average results of the k -th repetition of Monte-Carlo search

$$v_k(r, r, \dots, r) = \frac{1}{k+1}. \quad (3.24)$$

If first we repeat pure heuristics l times and afterwards Monte-Carlo search $m = k - l$ times, then

$$v_k(h, \dots, h, r, \dots, r) = \frac{1}{m+1}(1 - (1 - v(l, 0))^{m+1}). \quad (3.25)$$

If we start by repeating of Monte-Carlo search l times and afterwards the pure heuristics $m = k - l$ times, then

$$v_k(r, \dots, r, h, \dots, h) = \frac{1}{l+1}\left(\delta + \frac{1-\delta}{(m+1)^2}\right). \quad (3.26)$$

Now we consider a "mixture" of pure heuristics and of Monte Carlo search. Denote by q_i a probability of choosing Monte-Carlo search at the stage i , then the probability of pure heuristics at this stage will be $p_i = 1 - q_i$. Assume that probabilities q_i are independent. In this way we define probabilities of different (r, h) sequences. For example, the probability of Monte-Carlo sequence r, \dots, r is a product $\prod_{i=1}^k q_i$, and so on. Denote by $v_k(q)$ the average results given $q = (q_1, \dots, q_k)$.

Consider two simple cases for illustration.

Two-stage Case

For example, a pair (r, h) denotes that the Monte-Carlo search is used at the first stage and the pure heuristics at the second one. A symbol $v(r, h) = v_2(r, h)$ denotes average results given sequence (r, h) and a symbol $v(q) = v_2(q)$ defines average results given probability q .

Assume, for simplicity, that $q_1 = q_2 = q$. Then the average results given q

$$v(q) = q^2 v(r, r) + q(1 - q)(v(r, h) + v(h, r)) + (1 - q)^2 v(h, h). \quad (3.27)$$

The optimal value of $q = q^*$ is defined by the expression

$$q^* = \begin{cases} q^0, & \text{if } 2v(h, h) - (v(h, r) + v(r, h)) \geq 0, \\ & \text{and } 2v(r, r) - (v(h, r) + v(r, h)) \geq 0, \\ & \text{and } v(r, r) + v(h, h) - (v(h, r) + v(r, h)) > 0 \\ 0, & \text{if } 2v(h, h) - (v(h, r) + v(r, h)) < 0 \\ 1, & \text{if } 2v(r, r) - (v(h, r) + v(r, h)) < 0. \end{cases} \quad (3.28)$$

This expression was obtained by optimizing average $v(q)$ and meeting necessary constraints. Using expressions (3.23)(3.24)(3.25)(3.26) one may write that

$$q^0 = \frac{1}{2} \frac{81\delta^2 + 242\delta - 35}{81\delta^2 - 14\delta + 29}, \quad (3.29)$$

and

$$2v(h, h) - (v(h, r) + v(r, h)) < 0, \text{ if } 81\delta^2 + 242\delta - 35 < 0, \quad (3.30)$$

and

$$2v(r, r) - (v(h, r) + v(r, h)) < 0, \text{ if } 81\delta^2 - 270\delta + 93 > 0. \quad (3.31)$$

Solving (3.29)-(3.31) we obtain

$$q^* = \begin{cases} q^0, & \text{if } 0.138 \leq \delta \leq 0.390 \\ 0, & \text{if } \delta < 0.138 \\ 1, & \text{if } \delta > 0.390. \end{cases} \quad (3.32)$$

This means that pure heuristics are optimal, if the local minimum δ is close to the global one ($\delta < 0.138$). The "pure" Monte-Carlo search is optimal, if the local minimum is far away ($\delta > 0.390$). A mixture of pure heuristics and Monte-Carlo search is optimal if the parameter δ , that approximately defines the local minimum occurs between these limits.

Three-stage Case

In this case expression of average results $v(q) = v_3(q)$ given q is longer. Solving equations similar to expressions (3.29)-(3.31) we obtain

$$q^* = \begin{cases} q^0, & \text{if } 0. \leq \delta \leq 0.40 \\ 0, & \text{if } \delta < 0. \\ 1, & \text{if } \delta > 0.40. \end{cases} \quad (3.33)$$

Here

$$q^0 = \frac{A + \sqrt{A^2 - BC}}{B}, \quad (3.34)$$

where

$$A = 0.14\delta^3 - 3.91\delta^2 - 1.09\delta - 0.22, \quad (3.35)$$

and

$$B = -0.42\delta^3 + 6.15\delta^2 + 1.05\delta + 0.88 \quad (3.36)$$

and

$$C = 1.58\delta^2 - 0.19\delta + 0.11. \quad (3.37)$$

The examples show that using a mixture of pure heuristics and Monte-Carlo search one may obtain better results than using them separately. It seems natural since using a mixture we optimize a real variable $q \in [0, 1]$ instead of the Boolean one corresponding to the choice between the pure heuristics $q = 0$ and the Monte- Carlo search $q = 1$

3.4 "LEARNING" OF BHA

In pattern recognition systems "learning", as usual, is via "teaching" sequences. A "teacher" declares the true state and the recognizing system adapts its parameters accordingly.

Here we consider learning without a teacher. The system optimizes a sequence of problems and updates optimization parameters by the results. One may regard that as a "self-learning" or extrapolation. However we use a shorter term, namely, "learning".

Using BHA in a non-learning mode we optimize its parameters for each problem separately. If solving a new problem we use BHA parameters which were optimal while solving some other problems, then we work in a learning mode. A learning mode is useful while solving "on-line" problems when we are short of time for BHA parameter optimization. In such a case, we may use BHA parameters obtained from a "learning set" containing a number of real life or test problems. The "quality" of learning is defined as the difference between the "learned" parameters and the "actual"³ ones.

Denote by Ω a set of optimization problems

$$\min_{d \in D} v_\omega(d), \quad \omega \in \Omega. \quad (3.38)$$

Denote the optimal BHA parameters by x_ω (see Chapters 1 and 11).

$$x_\omega = \arg \min_{x \in X} f_\omega(x). \quad (3.39)$$

³By "actual" we mean optimal for a given problem.

Suppose that problems ω are generated independently and uniformly distributed in Ω . Assume that to each ω there corresponds one and only one optimal BHA parameter x_ω . Then expression (3.39) defines a mapping of the set Ω into the set X . Assume, for simplicity, the existence of probability density $p(x_\omega)$.

Denote by $\Omega_l(m) \subset \Omega$ a learning set of m problems. Suppose that all the m problems of the learning set $\Omega_l(m)$ are generated independently and uniformly distributed in Ω . Denote the learned parameters (parameters defined considering the learning set) by x_l . For example, one may define learned parameters as an averages on the learning set. Then

$$x_l = 1/m \sum_{\omega \in \Omega_l(m)} x_\omega. \quad (3.40)$$

Consider learned parameters x_l as some approximation of optimal parameters x_ω , $\omega \in \Omega \setminus \Omega_l(m)$, where index ω denotes the sample function to be optimized. Denote by $R(l, m)$ an expected error, when an approximate parameter x_l is used instead of the true one x_ω . We define it as learning error. Assume a quadratic loss function

$$l(x_\omega, x_l) = \|x_\omega - x_l\|^2. \quad (3.41)$$

Then, from expressions (3.40) and (3.41)

$$\begin{aligned} R(l, m) &= \mathbf{E}l(x_\omega, x_l) = \mathbf{E}(x_\omega - x_l)^2 \\ &= \mathbf{E}x_\omega^2 - 2\mathbf{E}x_\omega x_l + \mathbf{E}x_l^2. \end{aligned} \quad (3.42)$$

From the independence of problems $\omega \in \Omega$, it follows that x_ω are independent, too. Therefore

$$\mathbf{E}x_\omega x_l = \mathbf{E}x_\omega \mathbf{E}x_l \quad (3.43)$$

consequently

$$\begin{aligned} R(l, m) &= \mathbf{E}x_\omega^2 + 2\mathbf{E}x_\omega \mathbf{E}x_l + \mathbf{E}x_l^2 \\ &= \sigma_\omega^2 + \sigma_l^2 + (\mu_\omega - \mu_l)^2. \end{aligned} \quad (3.44)$$

Here $\mu_\omega = \mathbf{E}x_\omega$, $\mu_l = \mathbf{E}x_l$, $\sigma_\omega^2 = \mathbf{E}x_\omega^2 - (\mathbf{E}x_\omega)^2$, $\sigma_l^2 = \mathbf{E}x_l^2 - (\mathbf{E}x_l)^2$.

We generate learning problems independently in Ω , consequently it follows from (3.40) that

$$\begin{aligned} \mu_\omega &= \mu_l \\ \sigma_l^2 &= 1/m \sigma_\omega^2. \end{aligned} \quad (3.45)$$

From expressions (3.44) and (3.45) it follows that

$$R(l, m) = \frac{m+1}{m} \sigma_\omega^2. \quad (3.46)$$

We see that the learning error depends on the number m and the variance σ_ω . Given the number m , the learning error is determined by the variance σ_ω . Thus one may say that the maximal variance means no learning, and the minimal one means a complete learning. We say that learning is "weak", if the variance is equal to that of a uniform distribution of $x_\omega \in X$.

This definition of BHA learning is illustrated by a one-dimensional example, where $X = [-1/2, 1/2]$. In this case, weak learning error is

$$R(l, m) = (m+1)/m \int_{-1/2}^{1/2} x^2 dx = 1/12, \quad (3.47)$$

the maximal learning error (no learning) is

$$R(l, m) = (1/2(1/2)^2 + 1/2(-1/2)^2)(m+1)/m = 1/4 (m+1)/m, \quad (3.48)$$

and the minimal learning error (complete learning) is zero.

The complete learning means that the optimal parameters x_ω remain the same for all the problems $\omega \in \Omega$. The no-learning means that these parameters are as far away as possible from the learned value x_l . The weak learning corresponds to a uniform distribution of optimal parameters $x_\omega \in X$.

The following example illustrates a smooth change in the learning error. Suppose that the density of optimal parameter x_ω is

$$p(x_\omega) = \begin{cases} 1/\Delta, & \text{if } x_\omega \in [-\Delta/2, \Delta/2] \\ 0, & \text{otherwise} \end{cases}. \quad (3.49)$$

Then

$$R(l, m) = 1/12 (m+1)/m \Delta. \quad (3.50)$$

If $\Delta = 1$, then we obtain a weak learning, if $\Delta \rightarrow 0$, then we approach a complete learning, if $0 < \Delta < 1$, then the learning is between weak and complete.

The idea may be directly extended to a multi-dimensional case, too. We define a learning test by applying BHA to a set of uniformly distributed optimization problems. If the results of the test are uniform, that means a weak learning.

One may regard the learning good, if the results tend to concentrate in one area. For example, Figures 12.1 and 12.2 show that in the flow-shop example the learning of BHA is rather good because the density of optimal parameters is clearly non-uniform. The density is considerably greater around the point $x = (0.2, 0.5)$.

PART II

GLOBAL OPTIMIZATION

4

BAYESIAN APPROACH TO CONTINUOUS GLOBAL AND STOCHASTIC OPTIMIZATION

4.1 METHOD OF SEARCH

Consider a family C_A of continuous functions $f = f(x)$, $x \in A \subset R^m$. Assume a possibility to evaluate f at any fixed point x_n , $n = 1, \dots, N$, where N is the total number of observations.

The point of $n+1$ observation is defined by a decision function d_n in the following way: $x_{n+1} = d_n(z_n)$. Here the observed data are represented by a vector $z_n = (x_i, y_i, i = 1, \dots, n)$, $y_i = f(x_i)$. We represent the method of search by a vector $d = (d_0, \dots, d_N)$. Denote the final decision by $x_{N+1} = x_{N+1}(d)$ ¹. A deviation of the method d from the global minimum x^* is expressed as a linear function:

$$\delta = \delta(f, d) = f(x_{N+1}(d)) - f(x^*) \quad (4.1)$$

The worst case analysis corresponds to the "mini-max" condition:

$$\min_d \max_{f \in C_A} \delta(f, d) \quad (4.2)$$

The well known example of the mini-max method is a uniform grid, in a case of Lipschitz functions with an unknown Lipschitz constant (see [143]).

Define the average case analysis by the "Bayesian" condition:

$$\min_d \int_{C_A} \delta(f, d) dP(f) \quad (4.3)$$

¹Note that the index $N + 1$ and a reference to algorithm d defines the final decision made by algorithm d after N observations. A point of the last N -th observation is denoted as x_N . A point of the current observation is denoted by x_n , $n = 1, \dots, N$.

Here P is an a priori distribution.

4.2 DEFINING A PRIORI DISTRIBUTION

It is easy to see from expression (4.3) that Bayesian methods depend on an a priori distribution P . The choice of P is very wide. Therefore, first we must set some conditions that define "a proper family" of a priori distributions.

An a priori distribution is considered as "proper", if it ensures the convergence to the global minimum of any continuous function when n is large. The sufficient conditions (see [100]) are:

- the conditional variance converges to zero, if the distance to the nearest observation approaches zero.
- the conditional variance converges to some positive number, otherwise.

This means that we cannot predict exact values of an objective function outside the "densely observed" area. A large family of a priori distributions satisfies this condition. Some additional conditions should be introduced if we wish to narrow this family. Simple and natural are the three following conditions:

1. continuity of sample functions $f(x)$;
2. homogeneity of the a priori distribution P ;
3. independence of the m -th differences

Condition 2 means that a priori probabilities do not depend on the origin of coordinates. Condition 3 means that the m -th differences are like "white noise". Here the m -th differences can be regarded as discrete approximations of derivatives of the m -th order. Assumption 3 is the weakest condition compatible with a continuity of samples. It does not restrict the behavior of derivatives. As a result, the sample functions are non-differentiable almost everywhere.

The Gaussian a priori distribution with constant mean μ and the covariance function $\sigma_{j,k}$ satisfies these conditions if

$$\sigma_{j,k} = \prod_{i=1}^m \left(1 - \frac{|x_j^i - x_k^i|}{2} \right). \quad (4.4)$$

Here $x_j^i \in [-1, 1]$, $i = 1, \dots, m$. If $m = 1$, then a priori distribution (4.4) can be regarded as a sum of two Wiener processes running in the opposite directions plus a constant μ .

This example shows that the a priori distribution on a set of continuous functions is not so "subjective" after all. It can be derived from some simple and clear assumptions.

4.3 UPDATING A PRIORI DISTRIBUTIONS

Denote by $p_x(y)$ an a priori probability density of the objective function $y = f(x)$ at a fixed point x . By $p_x(y|z_n)$ denote a conditional probability density of $f(x)$ with regard to the observation vector z_n . It is referred to as "a posteriori density".

A transformation of $p_x(y)$ to $p_x(y|z_n)$ corresponds to the well known Bayesian formula. This transformation can be regarded as an updating of the a priori density after observing the results represented by the vector $z_n = (x_i, y_i, i = 1, \dots, n)$, $y_i = f(x_i)$.

In a Gaussian case, the a posteriori density can be expressed as:

$$p_x(y|z_n) = \frac{1}{\sqrt{2\pi}\sigma_n(x)} e^{-\frac{1}{2} \frac{(y - \mu_n(x))^2}{\sigma_n^2(x)}}. \quad (4.5)$$

Here $\mu_n(x)$ is the conditional expectation of $f(x)$ with regard to z_n . It shows expected values of $y = f(x)$ after n observations. The corresponding conditional variance is denoted by $\sigma_n^2(x)$. It defines the degree of uncertainty of $f(x)$ after n observations. Denote by $\mu(x)$ an a priori expected value of $f(x)$ and by $\sigma^2(x)$ the initial uncertainty represented as an a priori variance.

Denote by σ_{ij} an a priori covariance between $f(x_i)$ and $f(x_j)$. Denote by σ_{xi} an a priori covariance between $f(x)$ and $f(x_i)$. Here the corresponding covariance matrices are expressed as $\Sigma = (\sigma_{ij})$ and $\Sigma_x = (\sigma_{xi})$. Then the conditional expectation:

$$\mu_n(x) = \mu(x) + \Sigma_x \Sigma^{-1} (y - \mu(x)) \quad (4.6)$$

and the conditional variance:

$$\sigma_n^2(x) = \sigma^2(x) - \Sigma_x \Sigma^{-1} \Sigma_x^T \quad (4.7)$$

If the a priori distribution corresponds to equation (4.4) and $m = 1$, then expression (4.6) is a linear interpolation between the observed values y_i . Expression (4.7) defines a piecewise quadratic positive function with zero values at the observed points x_i . In this special case we need no inversion of the covariance matrix. The reason is that the a priori distribution corresponding to equation (4.4) is Markovian. This means that the conditional expectation and conditional variance depend only on two closest observations, one observation on the left and one on the right. It helps to design simple and efficient Bayesian methods for one-dimensional global optimization (see [148]). Unfortunately, the Markovian property holds only in a one-dimensional case.

4.4 DEFINING AND MINIMIZING RISK FUNCTIONS

It follows from condition 4.3 that a Bayesian method may be expressed in the following way:

$$d^* = \arg \min_d \int_{C_A} (f(x_{N+1}(d)) - f(x^*)) dP(f)$$

This condition can be simplified omitting the second component of the integral which does not depend on d . Thus we have:

$$d^* = \arg \min_d \int_{C_A} f(x_{N+1}(x)) dP(f) \quad (4.8)$$

Equation (4.8) can be reduced to an N-dimensional dynamic programming problem (see [99]). However, this problem is too difficult. Therefore one-step approximation is used. For any n we suppose that the $n + 1$ observation is the last one, and so on, until we reach $n = N$.

Assume P such that the minimum of the conditional expectation $\mu_{0n} = \min_{x \in A} \mu_n(x)$ is equal to the minimal value observed $y_{0n} = \min_{1 \leq i \leq n} y_i$, like in the Wiener process. Then the point of the next observation $x = x_{n+1}$ (see Figure 4.1) should minimize the conditional expectation of the least of the two values: $f(x)$ and c_n , where $f(x)$ is a predicted value at the point x and $c_n = y_{0n} - \epsilon_n$. Here ϵ_n is a correction parameter. This parameter can control the influence of the remaining observations. The parameter should be large if there are many of them. It should be small otherwise. In the one-step Gaussian case, the risk function can be expressed as:

$$R_n(x) = \frac{1}{\sqrt{2\pi}\sigma_n(x)} \int_{-\infty}^{+\infty} \min(y, c_n) e^{-\frac{(y-\mu_n(x))^2}{\sigma_n^2(x)}} dy \quad (4.9)$$

Consequently the Bayesian method can be defined as the following sequence:

$$x_{n+1} = \mathop{\text{arg min}}_{x \in A} R_n(x), n = 1, \dots, N. \quad (4.10)$$

We see from equations (4.6), (4.7), and (4.9) that to define the risk function for $m > 1$ one needs to inverse the covariance matrix of the n -th order. It is too difficult when n is large. Consequently one has no other choice than to abandon the classical framework of the probability theory. We shall replace some of its basic assumptions by other conditions more convenient for calculations.

The inversion of the covariance matrix Σ corresponds to the solution of the system of linear equations representing Kolmogorov's consistency conditions. Thus, the only way to avoid the expensive inversion is to omit those conditions.

It seems reasonable to replace Kolmogorov's consistency conditions by the following three assumptions:

1. continuity of risk function (4.9);
2. convergence of method (4.10) to the global minimum of any continuous function;
3. simplicity of expressions for $\mu_n(x)$ and $\sigma_n(x)$.

Then method (4.10) can be expressed (see [100]) as:

$$x_{n+1} = \mathop{\text{arg max}}_{x \in A} \min_{1 \leq i \leq n} \frac{\|x - x_i\|^2}{y_i - y_{0n} + \epsilon_n}, n = 1, \dots, N. \quad (4.11)$$

4.5 CONVERGENCE OF BAYESIAN METHODS

The main motivation for introducing the Bayesian search was not its asymptotic performance. We wanted to minimize an average deviation after some

finite (usually small) number of observations. In this way we defined Bayesian methods (4.10) and (4.11).

Let us consider these methods from the point-of-view of asymptotic behavior, too. We have already included the usual convergence conditions as condition 2 defining Bayesian method (4.11). This condition is weak. It does not show the efficiency of search. The efficiency of search can be defined as a relation of the density of observations in the area of global minimum to the corresponding average density. It is denoted by Γ_n . Then the limiting behavior of Γ_n for method (4.11) can be expressed as:

$$\Gamma = \lim_{n \rightarrow \infty} \Gamma_n = \frac{f_a - f_0 + \epsilon}{\epsilon}, \quad (4.12)$$

where

$$\epsilon = \lim_{n \rightarrow \infty} \epsilon_n.$$

Here $\epsilon_n > 0$ is a correction parameter, f_0 is a minimal value of $f(x)$ and f_a is an average value of $f(x)$ (see [100]),

One can see from equation (4.12) that the search will be nearly uniform asymptotically if the correction parameter ϵ is large, or if the objective function $f(x)$ is flat, what means that $f_a - f_0$ is small. If the correction parameter is small, then most of observations will be in the neighborhood of the global minimum asymptotically. However, a large number of observations will be needed to approach the asymptotic condition (4.12).

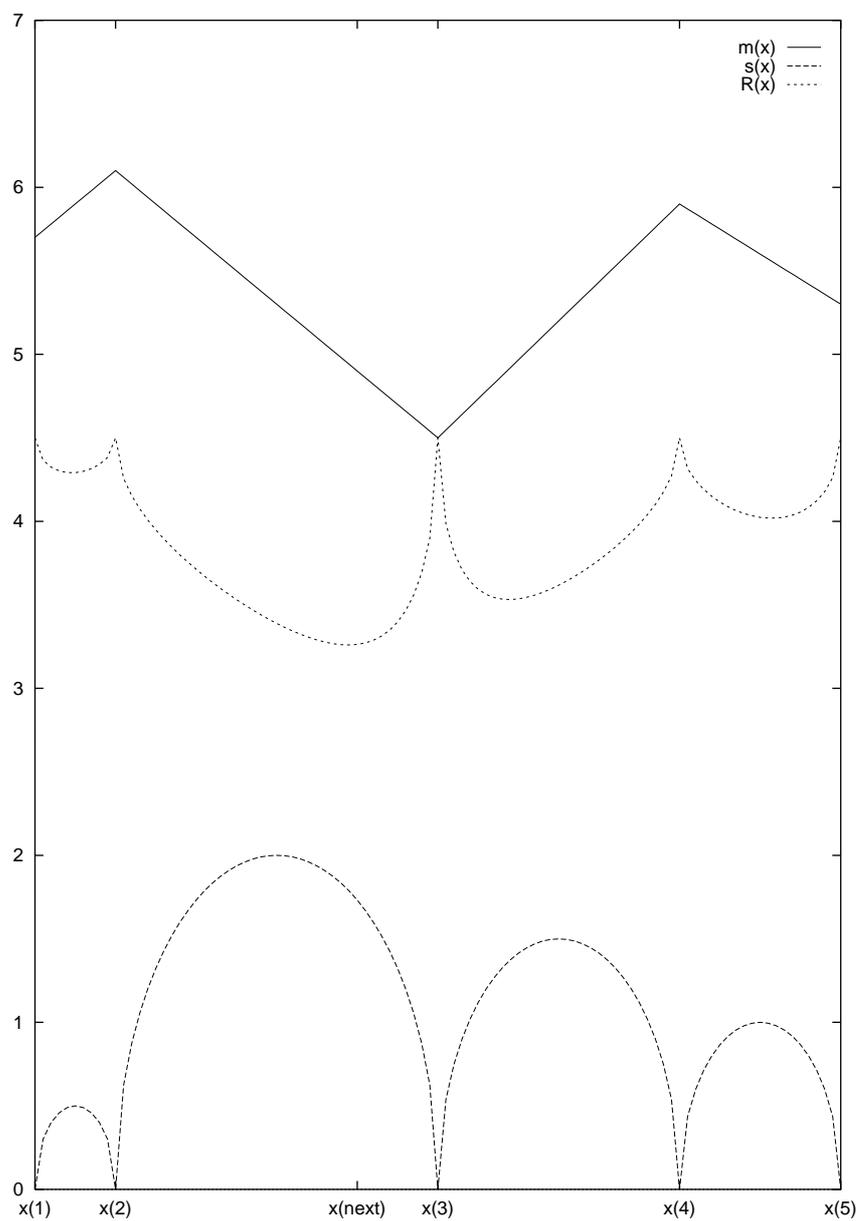


Figure 4.1 Conditional expectation $m(x)$, conditional standard $s(x)$, and risk function $R(x)$ with regard to fixed values $x(1), x(2), \dots$

5

EXAMPLES OF CONTINUOUS OPTIMIZATION

5.1 OUTLINE

Many examples of applications are related to the optimization of parameters of mathematical models represented as systems of non-linear differential equations. The objective function $f(x)$ depends on the solution of equations. Variables x represent controlled parameters of the system. The three following examples:

- maximization of the general yield of differential amplifiers (see Section 5.2),
- optimization of the mechanical system of shock-absorber (see Section 5.3),
- estimation of the parameters of non-linear regression of an immunological model (see Section 5.4),

belong to such a family of problems. The last example suggests a broad area for applications of global optimization. It is well known that in non-linear regression square deviation as well as a likelihood function could be multi-modal one for some data. The number of local minima may be very large, even in simple cases. An example is

- estimation of unknown parameters of bilinear time series (see Section 5.5).

A large "source" of difficult global optimization problems is engineering design. Here one optimizes parameters of mathematical models, usually non-linear. An example is

- optimization of composite laminates (see Section 5.6).

Many laws of nature could be defined in terms of global optimization. An example:

- the "Disk" problem: minimization of potential energy of organic molecules (see Section 5.7).

We are often not able to describe the behavior of new materials and technologies by mathematical models, because the corresponding information and knowledge is not available. Here we optimize by direct experiments, changing the control variables and observing the results. An example:

- The planning of extremal experiments of thermostable polymeric composition (see Section 5.8).

Let us now consider these examples separately.

5.2 MAXIMIZATION OF THE GENERAL YIELD OF DIFFERENTIAL AMPLIFIERS

In the production of Large Scale Integration (LSI) electronic circuits there are some inevitable deviations of dimensions of the LSI elements such as transistors and resistors from the ratings, set by a designer. As a result, some of the circuits do not meet the standards. The reason is that such parameters like delay time τ , a lower level of output voltage U or a bias of zero u may get out of the feasible region.

We define these parameters by a system of non-linear differential equations, depending on the dimensions of transistors and resistors. Usually the system of differential equations is solved using specific numerical techniques. Deviations of dimensions from the fixed ratings are simulated using some Monte Carlo techniques assuming a multi-variate Gaussian distribution.

In addition, there are "catastrophic" rejects, due to the defects of a silicon crystal.

Therefore the yield function can be expressed as a product of three components:

- The number of circuits in a crystal . It is a decreasing function of dimensions.
- The unit minus the fraction of rejects, attributable to the deviation of parameters. It is an increasing function of dimensions;
- The unit minus the fraction of rejects, due to crystal defects. It is a decreasing function of dimensions.

The product of monotonous functions is not necessarily unimodal. Simulating a differential amplifier we got the yield function that is two-modal for each variable representing the width or the length of a transistor. This means a possibility of 2^{2m} modality, where m is the number of transistors.

A multi-modality of the yield function together with the presence of noise, inevitable in any Monte-Carlo simulation, makes the problem very difficult. A coordinate optimization (global search optimizing each variable separately one-by-one (see Chapter 18)) seems convenient enough. It includes elements of visualization because one may see how the objective depends on separate dimensions during coordinate optimization.

The noise is filtered by Wiener smoothing. This means that the yield function is assumed to be a Wiener process and that the simulation error is a Gaussian noise with a standard deviation σ (see [7]). If σ is large, then one gets a horizontal line corresponding to the average value of observations. This means complete filtering and no optimization. If σ is zero, then we see a piece-wise linear line connecting the observed values. This means no filtering at all and a large number of pseudo local minima. The value $\sigma = 10$ is convenient for optimization and visual investigation of the yield function of a differential amplifier.

In one-dimensional global optimization the Wiener smoothing seems more convenient in comparison with the well known techniques of "smoothing of scatter-plots" (see [48]). In the Wiener smoothing there is only one parameter controlling the smoothing level, and this parameter has a clear statistical meaning. It can be regarded as the variance of Gaussian noise and can be estimated using the results of observations.

The idea of the Wiener smoothing can be regarded as a spline smoothing under some conditions (see [24]). The meaning of standard deviation of noise σ is similar to that of the spline smoothing parameter λ .

5.3 OPTIMIZATION OF THE MECHANICAL SYSTEM OF SHOCK-ABSORBER

Let us consider a mechanical object of two masses. Suppose that a shock is an instantaneous impulse and that a shock-absorber can be represented by a system of linear differential equations. The parameters of the shock-absorber should minimize the maximal deviation during a transitional process.

$$f(x) = \max_{0 \leq t \leq T} |\nu(t)|, \quad (5.1)$$

where $\nu(t)$ denotes a trajectory of the lower mass, and $f(x)$ means the maximal deviation during the transitional process. Four components of the vector $x \in B$ represent different relations of elasticities, masses and dampers. We define a feasible set B by non-linear constraints. Using penalty functions we can reduce the problem to the optimization in a rectangular set A , where $B \subset A$.

Expression (5.1) means that the objective $f(x)$ is defined by solving a one-dimensional multi-modal problem $\max_{t \in [0, T]} |\nu(t)|$. As a result, we get a four-dimensional multi-modal problem $\min_{x \in A} f(x)$.

A convenient way to maximize a one-dimensional multi-modal function $\nu(t)$ is by a relatively simple one-dimensional Bayesian method (see [164]). The four-dimensional function $f(x)$ is rather expensive. The calculation of this function for a fixed x is defined algorithmically and includes two procedures: the first one defining the trajectory $\nu(t)$, and the other maximizing $|\nu(t)|$. Thus when minimizing $f(x)$ we use the global multi-dimensional Bayesian method (see [100]).

The case of a non-linear shock-absorber (when we represent the object by a system of non-linear differential equations) can be treated in a similar way. The difference is that the numerical integration of non-linear differential equations should be applied instead of the analytic one. Another difference is that there appears an additional, fifth, parameter of "non-linearity."

5.4 ESTIMATION OF NON-LINEAR REGRESSION PARAMETERS OF THE IMMUNOLOGICAL MODEL

The well known mathematical model of immune response is a system of non-linear differential equations with time delay (see [9])

$$\begin{aligned}\frac{dv}{dt} &= -\gamma f v \\ \frac{df}{dt} &= \rho c - \eta \gamma f v - \mu_1 f \\ \frac{dc}{dt} &= \alpha f v|_{t-\tau} - \mu_2(c - c_0).\end{aligned}\tag{5.2}$$

Here $v = v(t)$ is a density of plasma antigen, $f = f(t)$ is a density of specific antibodies, $c = c(t)$ is the density of plasma cells, τ is time delay, and $\gamma, \rho, \eta, \mu_1, \mu_2, \alpha$ are unknown parameters of the model, represented by a vector x .

The relation between the parameters x of the model and the trajectories $v(t)$, $f(t)$ and $c(t)$ at the points t_i , $i = 1, \dots, K$ is defined by numerical methods. The objective function $f(x)$ is a likelihood function for some given data. Experimental data correspond to the reaction of a homogeneous sample of mice to the inoculation of a non-pathogenic antigen. The results of experiments are plasma cell densities at six fixed moments of time t_i , $i = 1, \dots, 6$.

For these data the likelihood function seems like a unimodal one. However, the global optimization turned out to be useful while selecting a good starting point for local optimization. That is important, because the efficiency of local search depends on the starting point.

The function $f(x)$ is not "expensive" due to the efficiency of specific numerical integration methods (see [9]). Thus the methods of uniform deterministic optimization proved to be sufficiently good in preliminary global search of the starting point for local optimization.

5.5 ESTIMATION OF BILINEAR TIME SERIES PARAMETERS

The class of bilinear time series models is useful for describing many non-linear phenomena. Let us consider a simple example:

$$y_i = x^1 y_{i-1} + x^2 y_{i-2} + x^3 y_{i-1} e_{i-1} + x^4 y_{i-2} e_{i-1} + e_i$$

Here $x = (x^1, \dots, x^4)$ are unknown parameters, y_i , $i = 1, \dots, k$ are experimental data, and e_i are residuals.

The way of behavior seen from this model, is rather common in seismological data (see [125]). For this type of data, the activity due to an event is of very short duration. The unknown parameters x can be estimated by minimizing the sum of residual squares:

$$f(x) = \sum_{i=1}^K e_i^2$$

It is easy to see that if i is large, then the residuals e_i are polynomials of a high degree, thus the multi-modality of $f(x)$ is almost inevitable. In that sense it is a good example of the application of global optimization. The function $f(x)$ is smooth, therefore some type of second order techniques such as variable metrics techniques may be used for local optimization.

Consider an example of bilinear time series (6.19) with $p = 2$, $q = 1$, $s = 2$, $r = 1$:

$$z_t - a_1 z_{t-1} a_2 z_{t-2} + \epsilon_t + c_{1,1} z_{t-1} \epsilon_{t-1} + c_{2,1} z_{t-2} \epsilon_{t-1}, \quad t = 1, \dots, T \quad (5.3)$$

For illustration we consider a simple case when $T = 12$, ϵ_t are Gaussian $\{0, 1\}$, and $a_1 = 0.8$, $a_2 = -0.4$, $c_{1,1} = 0.6$, $c_{2,1} = 0.7$. The mean square deviation $f(x) = \sum_{i=1}^{12} \epsilon_t$. Figure 5.1 shows how $f(x)$ depends on $c_{2,1}$. In this figure $d(x^4)$ denotes a mean square deviation $f(x)$, and x^4 denotes a variable parameter $c_{2,1}$.

5.6 OPTIMIZATION OF COMPOSITE LAMINATES

The optimization is usually most efficient in new developments. There are several reasons for that, such as the lack of experience, and the absence of

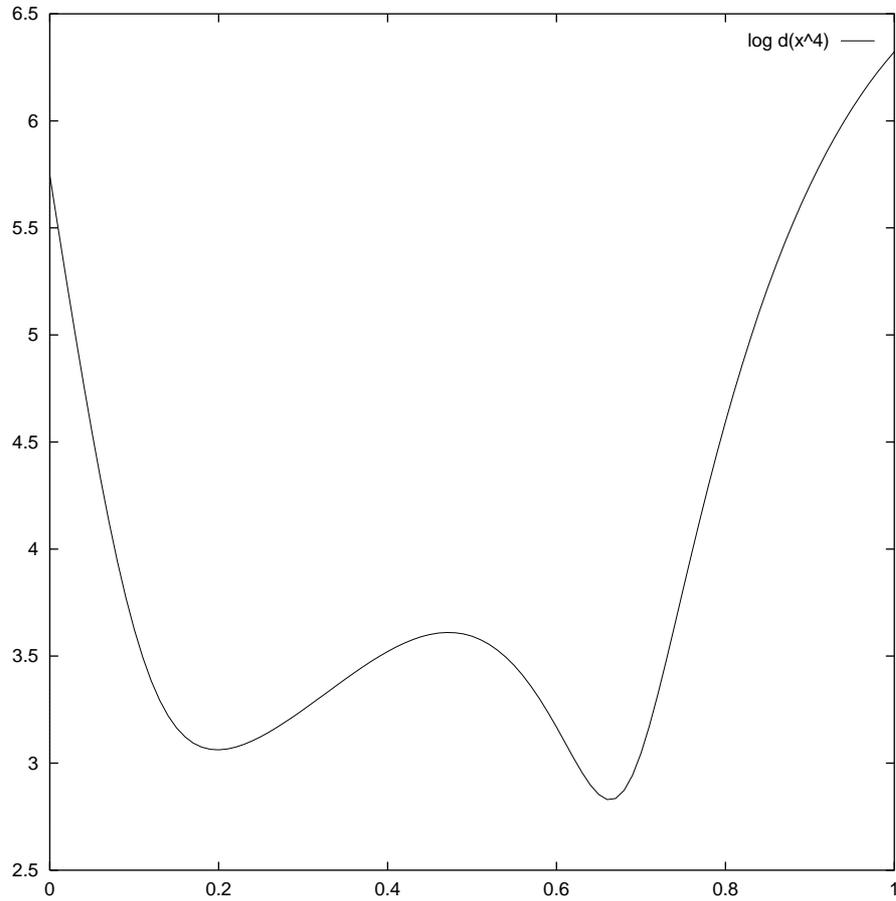


Figure 5.1 Bilinear Time Series

library of ready-made designs to be chosen. There is no "expert feeling" of the problem, which usually appears after some experience.

An example of a new application area is the design of laminated composite materials. Several thin layers (plies) are bound together to form a composite laminate. A single ply consists of long reinforcing fibers (e.g., graphite fibers), embedded within a relatively weak matrix material (e.g., epoxy). All fibers inside an individual ply are oriented in one direction. Composite laminates are usually fabricated in such a way that fiber angles vary from ply-to-ply.

The decision variable x^k is a fiber orientation angle in the k^{th} ply. Here k ranges from 1 to n and n equals the number of plies in the laminate. The angles x^i range from -90^0 to $+90^0$.

The objective function $f(x)$ is defined by experts as a sum of "penalty" functions corresponding to different plies $k = 1, \dots, n$ under different load conditions $j = 1, \dots, m$. It was supposed that penalties are exponentially increasing functions of calculated ply strains $\epsilon_{i(j)}^k$, $i = 1, 2$ and a ply shear strain $\gamma_{12(j)}^k$. It was assumed that the penalty functions are multiplied by a factor $\delta = m * n$, if the corresponding critical strains ϵ_i^{cr} and γ_{12}^{cr} are exceeded. Thus,

$$f(x) = \left(\frac{1}{m} \right) \sum_{j=1}^m \left[\sum_{k=1}^n \left[\left(\delta \cdot \exp \frac{|\epsilon_{1(j)}^k| - \epsilon_1^{cr}}{\epsilon_1^{cr}} \right)^2 + \left(\delta \cdot \exp \frac{|\epsilon_{2(j)}^k| - \epsilon_2^{cr}}{\epsilon_2^{cr}} \right)^2 + \left(\delta \cdot \exp \frac{|\gamma_{12(j)}^k| - \gamma_{12}^{cr}}{\gamma_{12}^{cr}} \right)^2 \right] \right] \quad (5.4)$$

The ply strains $\epsilon_{i(j)}^k$, $i = 1, 2$ and $\gamma_{12(j)}^k$ are functions of the decision variables x^k and the applied loadings. The summation is taken over all n -plies and m -loading conditions.

Three methods of global optimization were compared: Bayesian, uniform deterministic, and Hit-and-Run (see [162]). As expected, the Bayesian method used the available observations in the most efficient way. The method of Hit-and-Run consumed less computer time, due to simpler auxiliary calculations. The method of uniform deterministic search was somewhere between the Bayesian and the Hit-and-Run method, both in terms of computer time and the number of calculations.

This ordering of efficiency of the methods is natural for simple functions. We expect that with an increase in the complexity of $f(x)$, what apparently will happen taking into account many additional important factors, the Bayesian method may turn out to be the best one.

5.7 THE "DISK" PROBLEM: MINIMIZATION OF POTENTIAL ENERGY OF THE ORGANIC MOLECULE MODEL

Assume that the potential energy $f(x)$ of a molecule can be represented as a sum of functions $v_{ij}(r_{ij})$, where r_{ij} is the distance between atom i and atom j . Thus,

$$f(x) = \sum_{i,j=1,\dots,m,i<j} v_{ij},$$

where

$$v_{ij} = \left(\frac{s_{ij}}{r_{ij}}\right)^{12} - \left(\frac{s_{ij}}{r_{ij}}\right)^6.$$

Here

$s_{ij} = 1/2(s_i + s_j)$, s_j is a diameter of "atom" j and

$$r_{ij} = \sqrt{(x^i - x^j)^2 + (y^i - y^j)^2 + (z^i - z^j)^2}.$$

A vector (x^j, y^j, z^j) represents the center of atom j .

A two-dimensional problem was an attempt to compare different methods of global optimization at the CECAM Workshop on Global Optimization and Molecular Chemistry, Paris, June 1990. Most of the optimization methods at the Workshop were using also gradient values. It was supposed that the gradients were roughly $N + 1$ (where N is the number of variables) time more "expensive" and more "informative" in comparison with a single observation of $f(x)$. It was assumed that the minimal number of iterations in the seven point (14 variables) disk problem is ten. Consequently the minimal equivalent number of observations can be estimated as $10(14 + 1) = 150$.

The Bayesian algorithm was good in the sense of the number of observations. Some other methods such as tunneling (see [85]), stochastic equations (see [41]),

and a discrete dynamical system (see [31]), were more efficient with respect to computer time, due to simpler auxiliary calculations. The disk problem was considered in two stages.

In the first stage, the main objective is merely to get a preliminary understanding on the behavior of the objective function. In order to do that we would like to define some initial approximation to the global minimum, using a very small number of observations.

The advantage of this stage of research is that different methods can be easily compared using different computers, including PC. A disadvantage is that the bench-mark problem of 7 "atoms" is very small as compared to real organic molecules with 300 and more atoms. However, the results of comparison of the methods obtained for a small problem using a small number of observations may be useful for larger problems and, correspondingly, larger numbers of observations.

For the tasks of the first stage, a convenient tool is a PC and global optimization software with good graphics, for example, the system GM (see Chapter 19). The result after 9 observations of global Bayesian search and 108 observations of local search was -9.45 . It deviates significantly from the global minimum which is about -12.6 . One of the reasons of such a great deviation is that the number of observations was very small. Another reason is that global optimization methods are usually most efficient, if the objective function can be represented as a sum of two components: one unimodal and one multi-modal. The multi-modal component makes the sum also multi-modal. The projection of the potential energy function looks as a nearly constant function with occasional high and narrow spikes.

The second stage is to compare the results without restricting the observation number. The result after 153952 observations using the clustering algorithm was -12.5396 , what is quite near to the global minimum. Some other methods achieved a similar result, only the number of observations was larger.

5.8 PLANNING OF EXTREMAL EXPERIMENTS ON THERMOSTABLE POLYMERIC COMPOSITION

The objective function $f(x)$ is a specific loss of mass kg/m^2 in the flow of high temperature gases during a fixed interval (20 sec.).

There are four variables $x = (x^1, x^2, x^3, x^4)$, where

x^1 is the ratio of carbonized phenol- formaldehyde resin to phenol- formaldehyde,

x^2 is the percentage of urotropine,

x^3 is the specific pressure of moulding kg/cm^2 ,

x^4 is the time of moulding *min.*

Technological constraints are given in the form $a_i \leq x^i \leq b_i, i = 1, \dots, 4$.

No mathematical description of the objective function was available. The expert opinion was that $f(x)$, perhaps, has more than one local minimum. The only way to define the values of $f(x)$ at fixed points x_i was by physical experiment. In such a case, noise η is present as usual. The noise makes the testing of uni-modality of $f(x)$ very difficult. Thus, applying local methods of extremal experimental planning we can get stuck to some local minimum, which is far from the global one.

Table 5.1 Minimization of specific loss of mass

i	x_i^1	x_i^2	x_i^3	x_i^4	$f(x_i)$
1	50	13	4 070	8	2.141
2	55	14.5	6 990	12	0.737
4	65	12.5	5 210	4	0.514
5	70	11	5 909	30	0.495
11	70	14	3 289	3	0.493
8	77	7.7	3 700	20	0.417
26	80	10.5	4 320	17	0.385

Here the application of Bayesian methods seems natural. The first 12 observations were uniformly distributed using a deterministic uniform search (see [136, 141] and Sections 18, 19). The following 14 observations were carried out by the one-step Bayesian algorithm with the Gaussian a priori distribution defined by expression (4.4)¹.

Table 5.1 shows only the results of these observations when the quality of material was increasing. The best value of specific loss 0.385 was considered to be good for the materials of the given type at the time.

¹This algorithm is not in the GM software, because it satisfies Kolmogorov's consistency conditions and therefore needs inversion of the covariance matrix (see Section 4.4 and [100]), what is a very expensive operation, if the number of observations is large. The experimenters ignored the computing expenses because the cost of physical experimentation was very high.

6

LONG-MEMORY PROCESSES AND EXCHANGE RATE FORECASTING

6.1 INTRODUCTION

Modeling persistence in economic and financial time series using the autoregressive fractionally integrated moving average (ARFIMA) method has attracted the attention of many researchers in recent years [29, 16, 161, 17, 81, 103]. The frequency of the use of ARFIMA modeling in empirical research underscores the importance of efficient, both computational and statistical, estimation of the models. In estimating the parameters of the ARFIMA models, three approaches have been used: Maximum Likelihood (ML) [142], approximate ML [86, 46, 65, 66], and two-step procedures [51, 70]. Geweke and Porter-Hudak's method [51], unlike the ML approach, is less computationally demanding but some analysts consider it to be inadequate for finite samples.

In all the cases local optimization techniques were used. Theoretically we may apply the global optimization algorithms to the traditional ML problem. However, for parsimony in computation, the local optimization is usually applied [142]. In this case, the optimization results depend on the initial values, what implies that one cannot be sure if a global maximum is found. Furthermore, as noted in [21], the maximum likelihood estimation of ARFIMA models hinges upon estimations of the auto-covariances which are complicated functions of hyper-geometric functions. Thus in estimating auto-covariances of ARFIMA models one has to engage in computations of the hyper-geometric functions that may lead to serious round-off errors.

Theoretically, we may obtain a much better approximation to the global optimum using the advanced techniques of global optimization [63, 95, 148, 168].

However, the global optimization is very difficult in almost all the cases¹. The reason is a high complexity of multi-modal optimization problems in general. It is well known [79] that optimization of real functions cannot be done in polynomial-time, unless $P = NP^2$. In practice, this means that we need an algorithm of exponential time to obtain the ϵ -exact solution. The number of operations in exponential algorithms grows exponentially with the accuracy of solution and dimensions of the optimization problem (see Subsection 2.2.5).

A simple approximate approach in estimating the parameters of ARFIMA models is Least Squares (LS). We minimize the log-sum of square residuals instead of maximizing log-likelihood (see [103]).

Section 6.2 deals with the ARFIMA models and estimation methods. Section 6.5 considers the models of variable structure (VS) using the Iranian rial/\$ monthly "black" market exchange rate as an illustration. Section 6.6 investigates multi-modality problems using \$/£ and DM/\$ daily exchange rates, and AT&T and Intel Co stocks closing rates. Section 6.7 compares the average prediction results of ARMA and the Random Walk (RW) models.

6.2 AUTO-REGRESSION FRACTIONALLY-INTEGRATED MOVING-AVERAGE MODELS (ARFIMA)

6.2.1 Definitions

We define an ARFIMA³ process as a time series z_t ⁴

$$A(L)(1-L)^d z_t = B(L)\epsilon_t. \quad (6.1)$$

¹By 'difficult' we mean the time measure of computational complexity, that is, the minimum length of time would be needed for a standard universal computer to perform a task, see Subsection 2.2.5.

²The notation $P = NP$ means the existence a polynomial-time algorithm P for solving NP -complete problems. That is merely a theoretical possibility.

³Often the alternative title ARIMA(p,d,q) is used.

⁴Note that, contrary to the traditional practice, we do not assume the time series to be stationary. We allow the degree of integration of the series to be determined by direct estimation of the differencing parameter, d .

Here

$$A(L)w_t = w_t - \sum_{i=1}^p a_i w_{t-i} \quad (6.2)$$

and

$$B(L)\epsilon_t = \epsilon_t - \sum_{i=1}^q b_i \epsilon_{t-i}, \quad (6.3)$$

where $\epsilon_t = \text{Gaussian } \{0, \sigma^2\}$.

We define the transformation $(1 - L)^d$ as follows:

$$w_t = (1 - L)^d z_t = z_t - \sum_{i=1}^{\infty} d_i z_{t-i}. \quad (6.4)$$

Here

$$d_i = \frac{\Gamma(i - d)}{\Gamma(i + 1)\Gamma(-d)}, \quad (6.5)$$

where d is a fractional integration parameter, and $\Gamma(\cdot)$ is a gamma function.

We assume that

$$z_{t-i} = 0, \quad w_{t-i} = 0, \quad \epsilon_{t-i} = 0, \quad \text{if } t \leq i. \quad (6.6)$$

We truncate sequence (6.4)

$$d_i = 0, \quad \text{if } i > R. \quad (6.7)$$

Here R is the truncation parameter, the number of non-zero components.

6.2.2 Likelihood Maximization

The log-likelihood function of the ARIMA(p,d,g) process is defined as follows:

$$\log L(z; \mu, \Sigma) = -\frac{1}{2} \log |\Sigma| - \frac{1}{2} (z - \mu)^T \Sigma^{-1} (z - \mu). \quad (6.8)$$

Here Σ is a covariance matrix and μ is an expectation vector of z .

In [89] they used direct maximization of (6.8) to estimate the parameters of a fractional Gaussian model. To simplify the calculations in [65, 66] approximate optimization techniques were used, maximizing the log-likelihood function (6.8).

6.2.3 Minimization of Residuals

One of the advantages of residual minimization, as compared with the log-likelihood maximization, is that one may see directly how the objective depends on unknown parameters. We define residuals by recurrent expressions:

$$\begin{aligned}
 \epsilon_1 &= w_1 \\
 \epsilon_2 &= w_2 - a_1 w_1 + b_1 \epsilon_1 \\
 &\dots\dots\dots \\
 \epsilon_t &= w_t - a_1 w_{t-1} - \dots - a_p w_{t-p} + b_1 \epsilon_{t-1} + \dots + b_q \epsilon_{t-q}.
 \end{aligned}
 \tag{6.9}$$

Next the sum

$$f(x) = \log f_m(x), \quad f_m(x) = \sum_{t=1}^T \epsilon_t^2
 \tag{6.11}$$

is minimized.

The logarithm is used to decrease the objective variation by improving the scales. The objective $f_m(x)$ depends on $m = p + q + 1$ unknown parameters that are represented as an m -dimensional vector $x = (x_k, k = 1, \dots, m) = (a_i, i = 1, \dots, p, b_j, j = 1, \dots, q, d)$.

It is easy to see from (6.10), (6.4), and (6.2) that residuals ϵ_t are linear functions of the parameters a_t . This means that the minimum conditions

$$\frac{\partial f_m(x)}{\partial a_i} = 0, \quad i = 1, \dots, p
 \tag{6.12}$$

are given by a system of linear equations that defines the estimates of parameters $a_i = a_i(b, d)$ as a function of parameters $b_i, i = 1, \dots, q, d$. It reduces the number of parameters of non-linear optimization to $n = q + 1$.

The system

$$\frac{\partial f_m(x)}{\partial b_i} = 0, \quad i = 1, \dots, q
 \tag{6.13}$$

may have a multiple solution, because the residuals ϵ_t depend on b_i as polynomials of degree $T - 1$.

The equation

$$\frac{\partial f_m(x)}{\partial d} = 0 \quad (6.14)$$

may also have multiple solutions, because the residuals depend on d as a polynomial of degree R , where R is a truncation parameter.

These imply that, in general, the objective $f_m(x)$ is a multi modal function of parameters d and $b_i, i = 1, \dots, q$ ⁵. Therefore, one has to consider the methods of global optimization (see, for example [100]).

Denote

$$f(x) = \log f_{q+1}(x) \quad (6.15)$$

where

$$f_{q+1}(x) = f_m(x), \quad x_j = b_j, \quad j = 1, \dots, q, \quad x_{j+1} = d, \quad x_{j+1+i} = a_i(b, d), \quad i = 1, \dots, p.$$

this means that by condition (6.12) we define those x - components that represent the parameters $a_i, i = 1, \dots, p$.

There is no variance σ^2 in expressions (6.15) and (6.10). If necessary, we have to estimate the variance by some other well known techniques.

6.2.4 Advantages and Disadvantages of Squared Residuals Minimization (SRM)

An advantage of the SRM approach is that objective (6.11) is explicitly expressed as a function of variables a, b , and d . It is important in the investigation of multi-modality problems while looking for an efficient method of global optimization. Model (6.11) is flexible. For example, one may conveniently consider the sum of ARIFMA model with some other models,

A disadvantage of model (6.11) is a truncation (see expression (6.7)), and a change made by truncation in variances and covariances. Compare, for example, the simulated series of [56] which did truncate and that of [51] which did not.

We think that in some cases the advantages of model (6.11) outweigh the disadvantages. In any case, a simple model (6.11) is a good test problem while

⁵The same reasoning applies to the log-likelihood function, too.

developing some global optimization techniques that may be used for likelihood maximization, too.

For Box-Jenkins ARIMA models (see [12] and expressions (6.2) and (6.3)), the parameters a_i and b_j are estimated by formal methods. The parameter d is determined by intuitive examination of various plots of autocorrelation and partial autocorrelations of the estimated residuals obtained from expression (6.1). As a general rule fractional ARIMA processes with $0 < d < 1/2$ exhibit long memory, and for $d \geq 1/2$, the process is non-stationary.

Using the formal methods of estimation, such as the present one, we directly estimate the differencing parameter d as well as other parameters. Note that the ARFIMA model does not represent the long run relationships per se; although it detects persistence in the series if one exists. One can use the ARFIMA model to test for existence of long-run relationship between the variables of the model, but that method has not been elaborated in this book.

6.3 ARTIFICIAL NEURAL NETWORKS MODELS (ANN)

One uses the ARFIMA model (6.1) because of two reasons

- the model may detect the persistence in the time series;
- the model is linear regarding the data and non-linear as a function of parameters of both the moving average b and the fractionally integration d .

If we are interested mainly in the non-linearities, then we may apply many other non-linear models, including the ones that are non-linear regarding the data, too. In this book we will discuss two of them. In this section the ANN model will be considered. In the next section, we shall introduce a bilinear term into the ARFIMA model.

We apply ANN by involving the non-linear activation function ϕ (see (Section 10.1)) into the standard Auto-Regression model

$$w_t = \phi\left(\sum_{i=1}^p a_i w_{t-i}\right) + \epsilon_t. \quad (6.16)$$

The idea lurking behind ANN is that the activation function ϕ roughly represents the activation of a real neuron. We minimize the sum

$$f_m(x) = \sum_{t=1}^T \epsilon_t^2, \quad (6.17)$$

where the objective $f_m(x)$ depends on p unknown parameters represented as a p -dimensional vector $x = (x_k, k = 1, \dots, p) = (a_i, i = 1, \dots, p)$.

From expression (6.16) it is clear that the residuals ϵ_t are nonlinear functions of parameters a_t if the activation function ϕ is nonlinear⁶. This means that the minimum conditions

$$\frac{\partial f_m(x)}{\partial a_i} = 0, \quad i = 1, \dots, p \quad (6.18)$$

is a system of nonlinear equations that may have a multiple solution. Obviously, using ANN one obtains the same multi-modality problem as in a case of AR-FIMA model (see non-linear equations (6.13)). The multi-modality problems of ANN models are discussed in Chapter 10.

6.4 BILINEAR MODELS

It is well known that for the adequate description of some phenomena additional non-linear terms of the time series could be of use. A simple example is to add a bilinear term (see [125, 88]). Here are bilinear time series extending the ARFIMA model:

$$A(L)z_t = B(L)\epsilon_t + C(L)z_t\epsilon_t, \quad (6.19)$$

where

$$C(L)z_t\epsilon_t = \sum_{i=1}^s \sum_{j=1}^r c_{ij} z_{t-i} \epsilon_{t-j}. \quad (6.20)$$

For an illustrative example of the bilinear time series see Section 5.5.

⁶Assuming the linear activation function ϕ the ANN model is reduced to the standard Auto-Regression model.

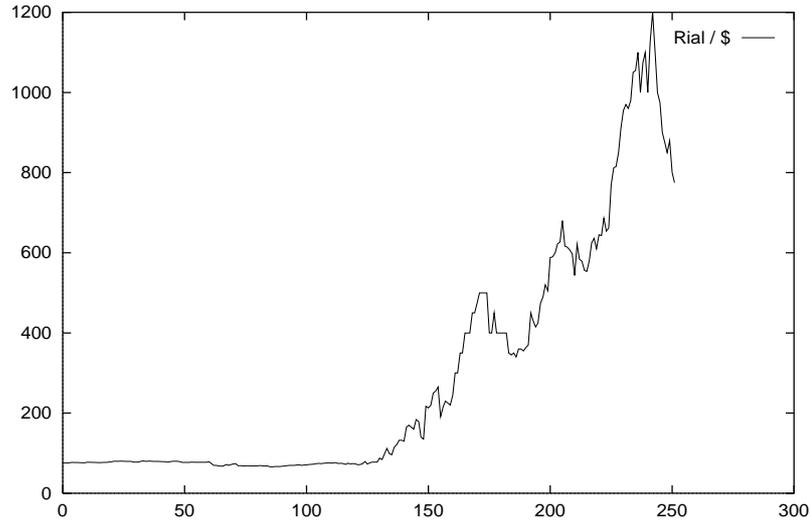


Figure 6.1 The rial/\$ monthly exchange rate in "black" market (1965-1988)

6.5 VARIABLE STRUCTURE MODELS (VSM)

There are cases when the model structure changes. An examples is the former Eastern-Block countries and some Developing Nations. A simple example illustrating how the proposed approach works in the VSM is rial/\$ monthly "black" market exchange rate. Assume that the real life financial models may be represented as a sum of two components: deterministic and stochastic.

6.5.1 Deterministic Component

Figure 6.1 shows the rial/\$ rate y_t as a function of time t . The series may be represented as a sum of a deterministic component μ_t and some ARIFMA process. Assume that

$$\mu_t = \begin{cases} \mu_0, & \text{if } t \leq T_0 \\ \mu_0 + \mu(t - T_0), & \text{otherwise.} \end{cases} \quad (6.21)$$

Here μ_0 and μ are unknown parameters, and T_0 is the structural change moment (the time of Iranian revolution). Note that the standard σ_t of the rial/\$ rate y_t

depends on μ_t . Suppose that

$$\sigma_t = \begin{cases} \sigma\mu_0, & \text{if } t \leq T_0 \\ \sigma(\mu_0 + \mu(t - T_0)), & \text{otherwise.} \end{cases} \quad (6.22)$$

Assume the residuals ϵ_t , as defined by expressions (6.10) and (6.4), to be independent with zero expectation and a constant standard σ

$$\begin{aligned} E\epsilon_t &= 0, \\ E\epsilon^2 &= \sigma^2. \end{aligned} \quad (6.23)$$

This assumption and conditions (6.21),(6.22), (6.10), and (6.4) hold, if

$$z_t = \frac{y_t - \mu_t}{\mu_t}. \quad (6.24)$$

6.5.2 Estimation of the Deterministic Model

Consider some simple estimators of deterministic parameters μ , σ , μ_0 . First we estimate the parameters μ_0 and σ using only $y_t, t \leq T_0$. It follows that

$$\bar{\mu}_0 = 1/T_0 \sum_1^{T_0} y_t, \quad (6.25)$$

$$\bar{\sigma}^2 = 1/T_0 \sum_1^{T_0} (y_t - \bar{\mu}_t)^2.$$

The parameter μ is estimated by minimizing the sum h of squared z_t defined by expression (6.24)

$$h(\mu) = \sum_{t=1}^T z_t^2. \quad (6.26)$$

It follows from (6.24) and the condition $dh(\mu)/d\mu = 0$ that

$$\sum_{t=1}^T \frac{(y_t^2(t - T_0) - y_t(t - T_0)(\mu_0 + \mu(t - T_0)))}{(\mu(t - T_0) + \mu_0)^3} = 0, \quad (6.27)$$

or

$$\sum_{t=1}^T \frac{(y_t^2(t - T_0) - y_t(t - T_0)(\mu_0 + \mu(t - T_0)))}{(\mu(t - T_0) + \mu_0/\mu)^3} = 0, \quad (6.28)$$

or

$$\sum_{t=1}^T \frac{(y_t^2(t-T_0))}{(\mu(t-T_0) + \mu_0/\mu)^3} = \mu \sum_{t=1}^T \frac{y_t(t-T_0)}{(\mu(t-T_0) + \mu_0/\mu)^3}. \quad (6.29)$$

After some rearrangement

$$\mu = \frac{\sum_{t=1}^T \frac{y_t^2(t-T_0) - \mu_0}{(\mu_0/\mu + (t-T_0))^3}}{\sum_{t=1}^T \frac{y_t(t-T_0)^2}{(\mu_0/\mu + (t-T_0))^3}}. \quad (6.30)$$

We may use the following simple estimate of μ as a first approximation

$$\tilde{\mu} = \frac{\sum_{t=T_0+1}^T (y_t - \mu_0)}{\sum_{t=T_0+1}^T (t - T_0)}. \quad (6.31)$$

The estimate $\tilde{\mu}$ is unbiased, because

$$E\tilde{\mu} = \frac{\sum_{t=T_0+1}^T E(y_t - \mu_0)}{\sum_{t=T_0+1}^T (t - T_0)} = \quad (6.32)$$

$$\frac{\sum_{t=T_0+1}^T (\mu(t - T_0) + \mu_0 - \mu_0)}{\sum_{t=T_0+1}^T (t - T_0)} = \mu. \quad (6.33)$$

To improve the estimate of μ one may use the iterative procedure. At the first iteration we obtain $\bar{\mu}$ from (6.30), by fixing $\mu = \tilde{\mu}$ on the right side of (6.30)

$$\bar{\mu} = \frac{\sum_{t=1}^T \frac{y_t^2(t-T_0) - \mu_0}{(\mu_0/\bar{\mu} + (t-T_0))^3}}{\sum_{t=1}^T \frac{y_t(t-T_0)^2}{(\mu_0/\bar{\mu} + (t-T_0))^3}}. \quad (6.34)$$

At the second iteration we replace $\tilde{\mu}$ by the estimate $\bar{\mu}$ and obtain an "updated" estimate $\bar{\bar{\mu}}$. The procedure is repeated until some stopping condition is met.

As an alternative, one may use the Newton method to minimize $h(\mu)$

$$\mu_{n+1} = \mu_n - s_n, n = 1, 2, \dots \quad (6.35)$$

where

$$s_n = dh(\mu_n)/d\mu \quad d^2h(\mu_n)/d\mu^2. \quad (6.36)$$

Here

$$d^2h(\mu_n)/d\mu^2 = \sum_{t=1}^T y_t(t-T_0)^2 \left(\frac{2}{(\mu_0 + \mu(t-T_0))^3} - \frac{3y_t}{(\mu(t-T_0) + \mu_0)^4} \right). \quad (6.37)$$

One may choose the iterative (6.34) or the Newtonian (6.35) technique depending on the initial choice of the parameters $\tilde{\mu}_0$ and $\tilde{\mu}$ that determines the convergence conditions.

6.5.3 Estimation of a Joint Variable Structure Model

We joined two different models to make the estimation of parameters μ_0, μ, σ more convenient. This helped us to "clean up" the ARIFMA models by reducing the influence of a deterministic component. ARIFMA parameters a, b, d are estimated before and after the Iranian revolution separately, using the common estimates $\bar{\mu}_0, \bar{\mu}, \bar{\sigma}$ for both ARIFMA models: $1 \leq t \leq T_0$ and $T_0 + 1 \leq t \leq T$. These models are referred to as model 1 and model 2.

Figure 6.2 shows the rial/\$ rate transformed by expression (6.24) for model 1 (from 1965 to 1976). For a better "visibility" we add the value of μ_0 to the data.

Figure 6.3 shows the rial/\$ rate transformed by expression (6.24) for model 2 (from 1977 to 1988). Estimating unknown ARIFMA parameters we minimize the log-sum of squared residuals defined by expression (6.11).

6.5.4 Multi-Modality in VSM

A set of figures 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10, 6.11 shows how log-sum (6.11) depends on the parameters b_0, b_1 , and d , correspondingly. The first four figures indicate the results of model 1. The remaining four show the results of model 2.

The basis of selecting intervals such as $b_0 \in [-0.5, 0.5]$, $b_0 \in [-0.35, 0.5]$, etc. is the proper scales. For example, selecting the interval $[-0.5, 0.5]$ we see the multi-modality. The objective looks flat in the region $[-0.35, 0.5]$. Selecting $[-0.35, 0.5]$ we see how the objective changes in this region, while the multi-modality "disappears". Parameters a are estimated by expression (6.12) directly. Inspecting Figure 6.4 we notice two distinct regions: a "multi-modal" region, and a "uni-modal" one. For better visualization we show only the "uni-modal" region in Figure 6.5.

Figure 6.6 shows that the relations of log-sum (6.11) on b_1 are more regular, meanwhile being multi-modal. Figure 6.7 shows that the relation of log-sum (6.11) on d is multi-modal. The global minimum of objective (6.11) as a function of d is at $d = 0.74$ ⁷. The local minimum is at zero.

⁷That is the only observed case when the optimal d is non-zero.

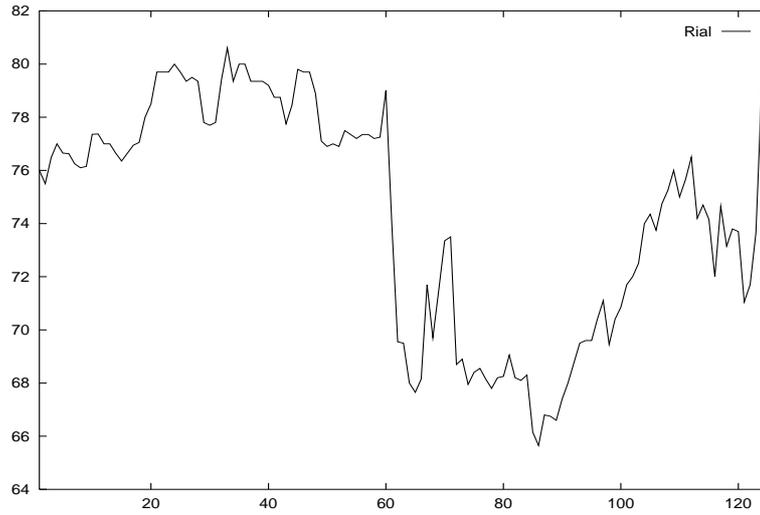


Figure 6.2 Transformed rial/\$ monthly exchange rate (1965-1976)

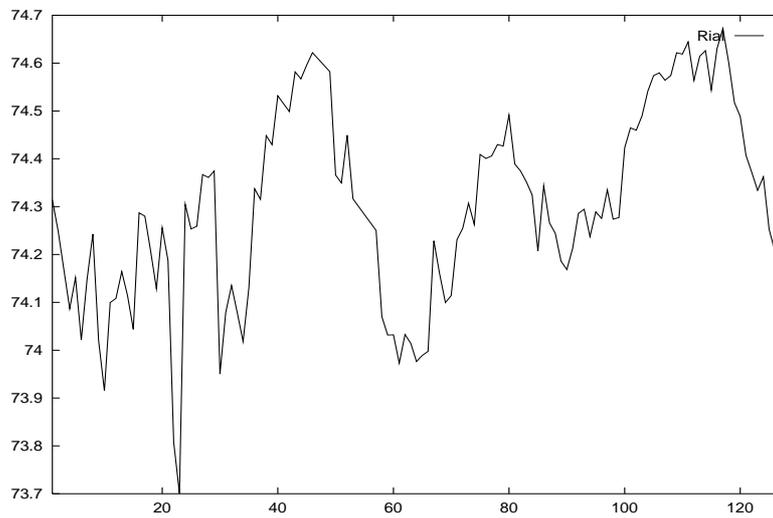


Figure 6.3 Transformed rial/\$ monthly exchange rate (1977-1988)

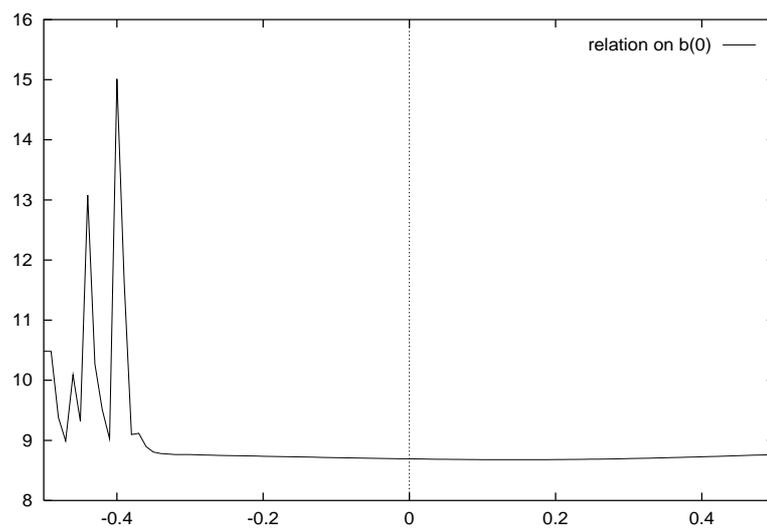


Figure 6.4 Log-sum (6.11) as a function of the parameter $b_0 \in [-0.5, 0.5]$ of the model 1

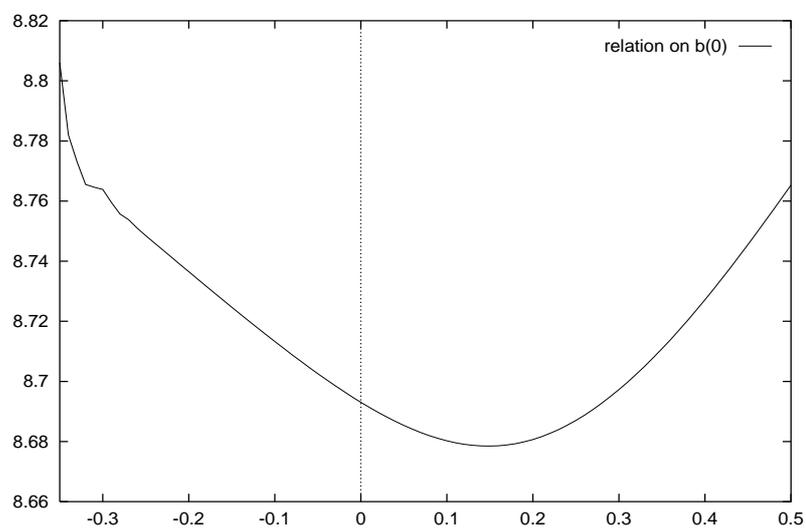


Figure 6.5 Log-sum (6.11) as a function of the parameter $b_0 \in [-0.35, 0.5]$ of the model 1

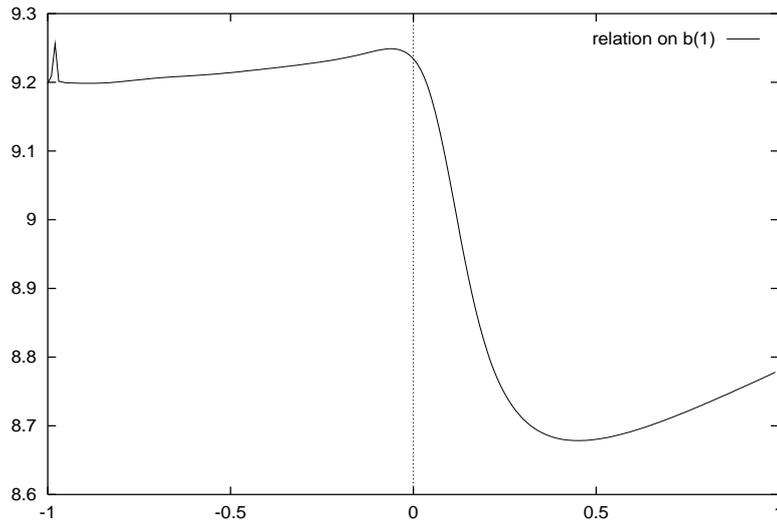


Figure 6.6 Log-sum (6.11) as a function of the parameter $b_1 \in [-1, 1]$ of the model 1

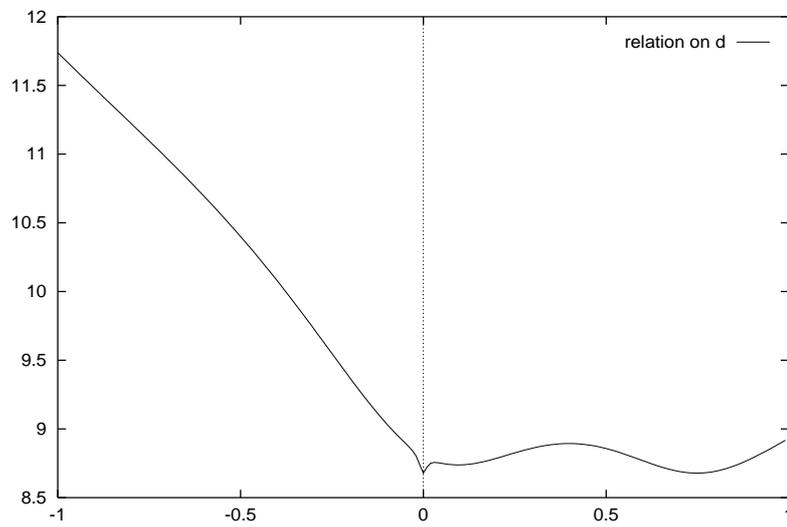


Figure 6.7 Log-sum (6.11) as a function of the parameter $d \in [-1, 1]$ of the model 1

Figure 6.8 shows a multi-modality of log-sum (6.11) as a function of b_0 in model 2. Figure 6.9 shows a "multi-modal" region of b_1 in model 2. Figure 6.10 illustrates only the "uni-modal" region. Figure 6.11 demonstrates that the global minimum of log-sum (6.11) as a function of d is exactly at zero, as usual.

The results witness that ignoring the multi-modality of sum (6.11) one cannot obtain reliable estimates of parameters b and d . Therefore the global optimization is needed.

6.5.5 Semi-Monte Carlo Simulation

A simple way of visual "validation" of a model is by Monte-Carlo simulation. The objective of simulation is to compare the real data with the simulated results of the statistical model defined by expressions (6.21), (6.22), and (6.4).

An obvious way to do that is by using some type of "Semi-Monte Carlo" simulation technique defined in short as "Simulated Forecasting" (SF). Using SF we fix the estimated values of unknown parameters μ_0, μ, σ common for both the models 1 and 2. We also fix the "individual" parameters a, b, d for each of two ARFIMA models.

The residuals ϵ_t (see expression (6.10)) are determined up to the simulation starting moment $t(s)$ using the observed data. The rest of residuals $\epsilon_t, t \geq t(s)$ are generated by a Gaussian distribution with zero mean and variance σ^2 . We repeat the simulation 10 times for each ARIFMA model separately. The results are presented in Figures 6.12 and 6.13.

The lines denoted as "real" show the real data (the rial/\$ exchange rate transformed by (6.24) plus μ_0). The "mean" lines show the average results of SF prediction of $y_t + \mu_0$. The "min" and "max" lines denote the lower and the upper values of simulation. Therefore, these lines are referred to as "SF- confidence intervals", meaning that if the model is true, one may expect those "intervals" to cover the real data with some "SF-confidence level" $\alpha(SF)$. It is very difficult to define $\alpha(SF)$ exactly. Assuming that "interval deviations" may be regarded as independent and uniformly distributed random variables, we obtain $\alpha(SF) = 1 - k^{-1}$. Here k is the number of Monte-Carlo repetitions. In our case, $k = 10$, thus $\alpha(SF) = 0.9$

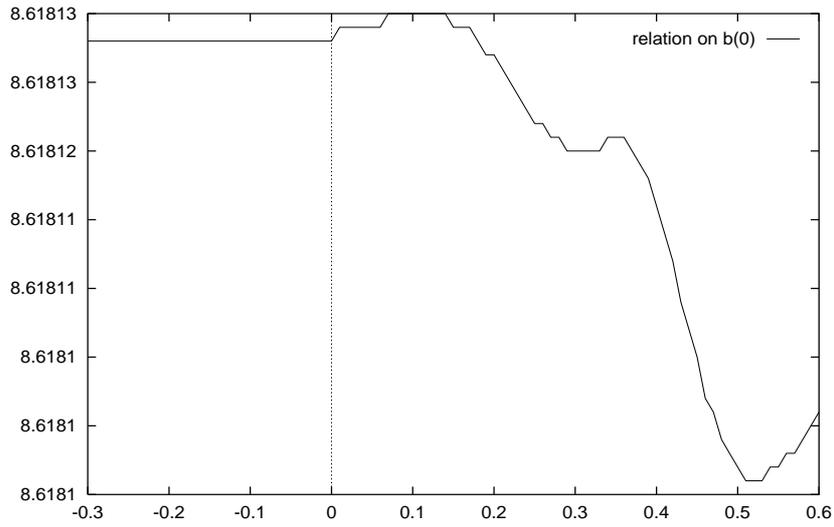


Figure 6.8 Log-sum (6.11) as a function of the parameter $b_0 \in [-0.3, 0.6]$ of the model 2

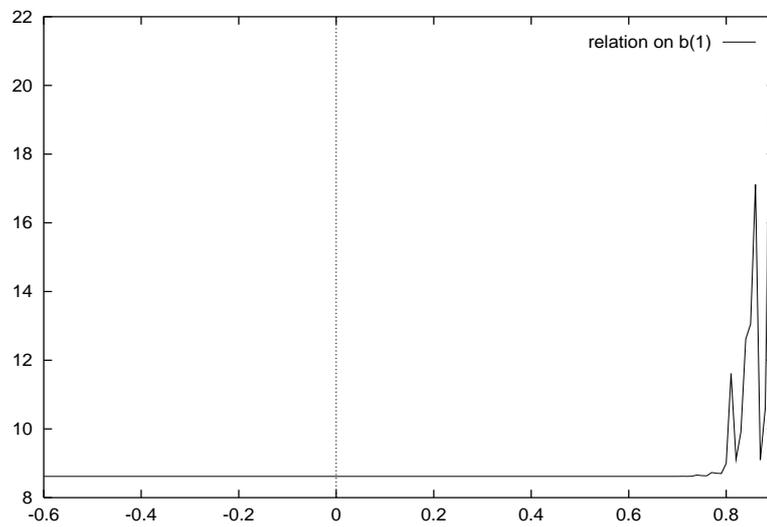


Figure 6.9 Log-sum (6.11) as a function of the parameter $b_1 \in [-0.6, 0.9]$ of the model 2

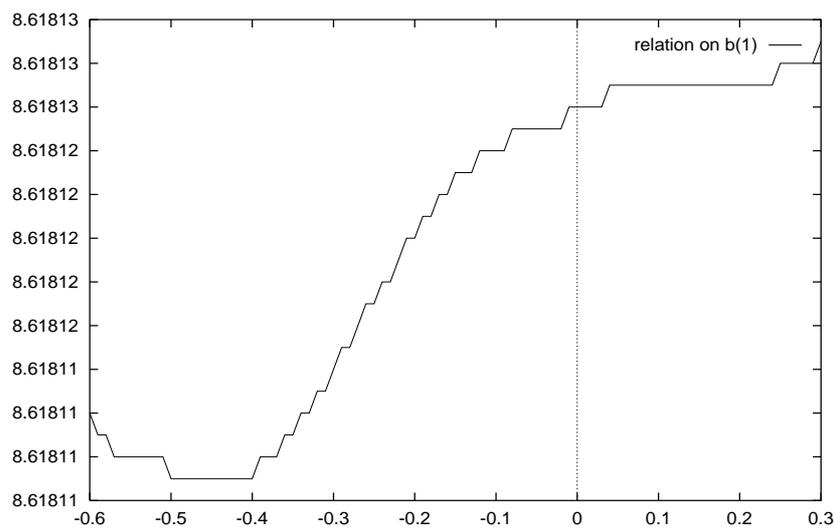


Figure 6.10 Log-sum (6.11) as a function of the parameter $b_1 \in [-0.6, 0.3]$ of the model 2

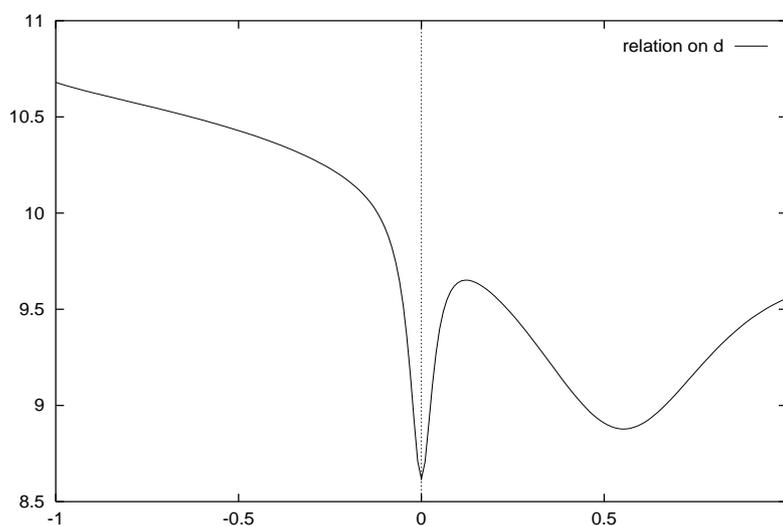


Figure 6.11 Log-sum (6.11) as a function of the parameter $d \in [-1, 1]$ of the model 2

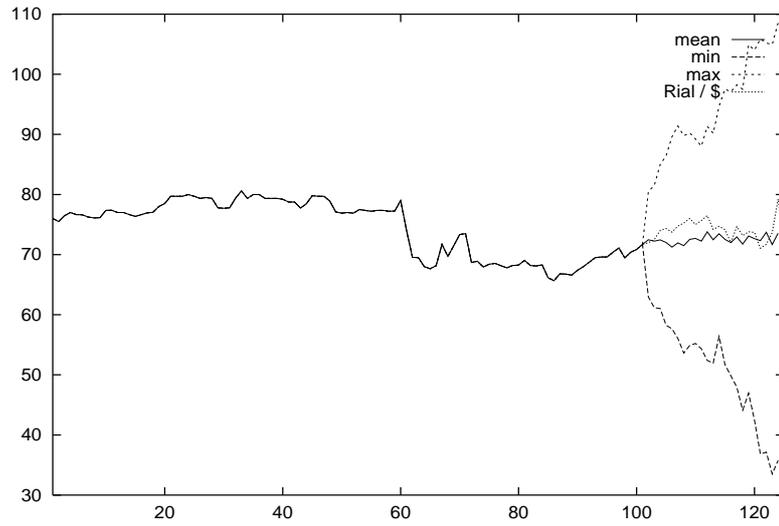


Figure 6.12 Results of Semi-Monte Carlo Simulation of the transformed rial/\$ monthly exchange rate (1965-1976)

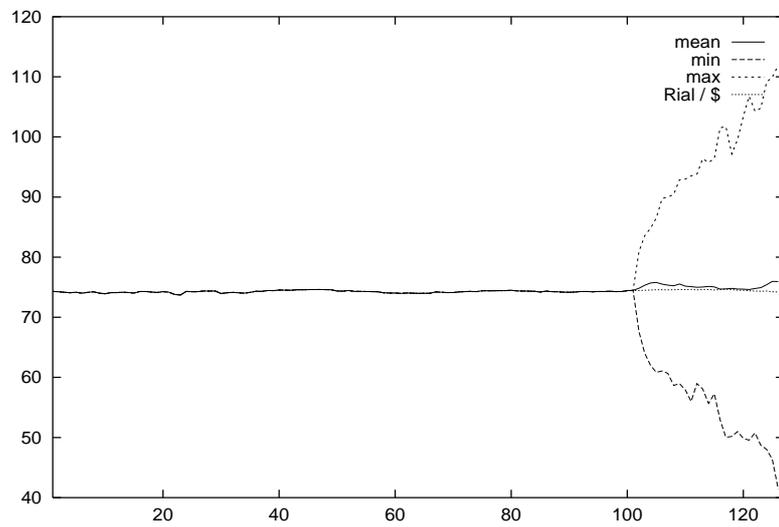


Figure 6.13 Results of Semi-Monte Carlo Simulation of the transformed rial/\$ monthly exchange rate (1977-1988)

Unfortunately, this assumption "over-simplifies" the statistical model. Therefore, one may regard "SF-confidence levels" $\alpha(SF)$ merely as a Monte-Carlo approximation.

6.6 OPTIMIZATION OF ARFIMA MODELS

6.6.1 Multi-Modality Examples

We consider $\$/\pounds$ and DM/\$ daily exchange rates and AT&T and Intel Corporation stocks closing rates as examples (see 6.14, 6.15, 6.16, 6.17). Estimating unknown ARFIMA parameters we minimize a log-sum of squared residuals defined by expression (6.11).

A set of figures 6.18, 6.19, and 6.20 shows how log-sum (6.11) depends on the parameters b_0 , b_1 , and d , considering the $\$/\pounds$ exchange rate.

A set of figures 6.21, 6.22, and 6.23 shows the same relation considering the DM\$ exchange rate.

A set of figures 6.24, 6.25, and 6.26 show the relation regarding the AT&T stocks closing rate.

A set of figures 6.27, 6.28, and 6.29 show the relation obtained from the Intel Corporation closing rate.

Parameters a are estimated by expression (6.12).

The figures indicate the multi-modality of log-sum (6.11) as a function of parameters b_0, b_1, d in all the three cases⁸. We see that zero is always the local minimum of objective (6.11) as a function of d . The point $d = 0$ is the global minimum, too, in most cases, an exception is model 1 (see Figure 6.7).

⁸That means a multi-modality of sum (6.11), too.

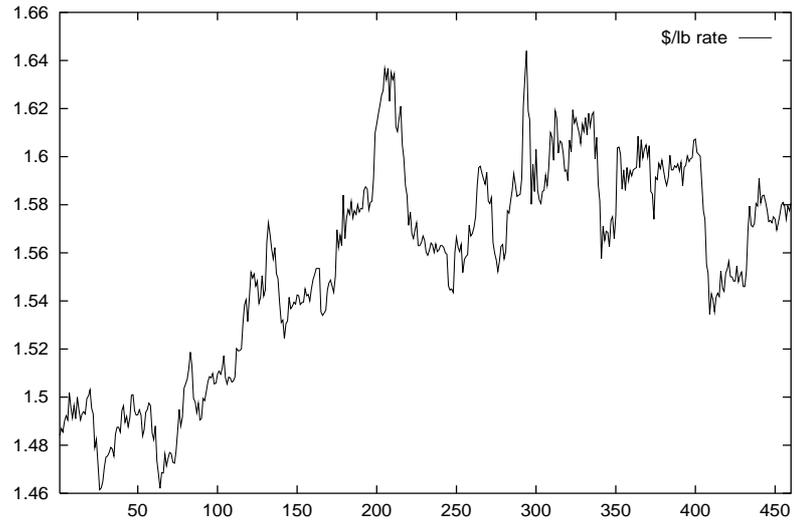


Figure 6.14 \$/£ daily exchange rate (starting from September 13, 1993)

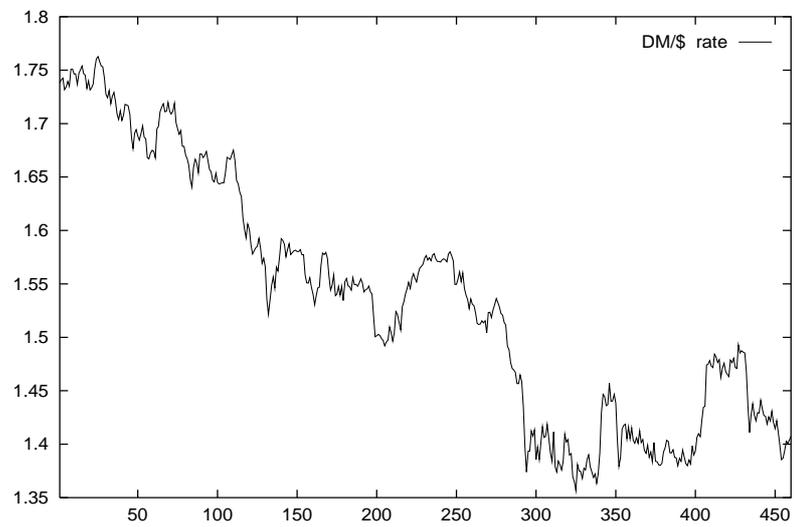


Figure 6.15 DM/\$ daily exchange rate (starting from September 13, 1993)

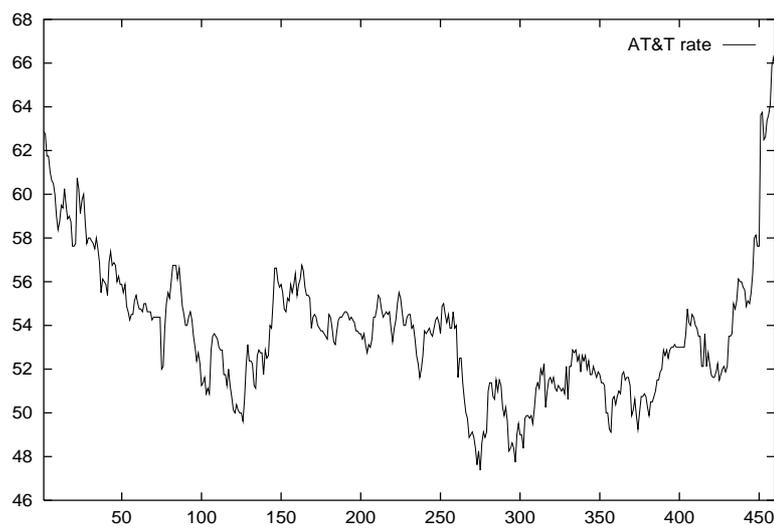


Figure 6.16 AT&T stocks closing rate (starting from August 30, 1993)

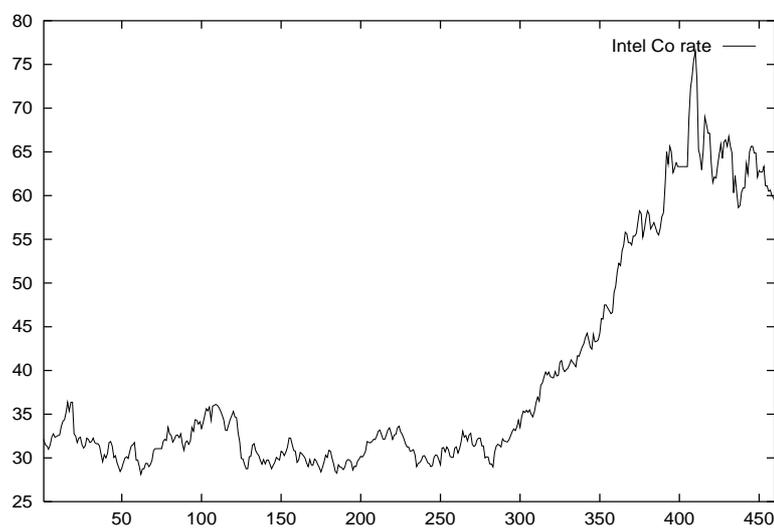


Figure 6.17 Intel Co. stocks closing rate (starting from August 30, 1993)

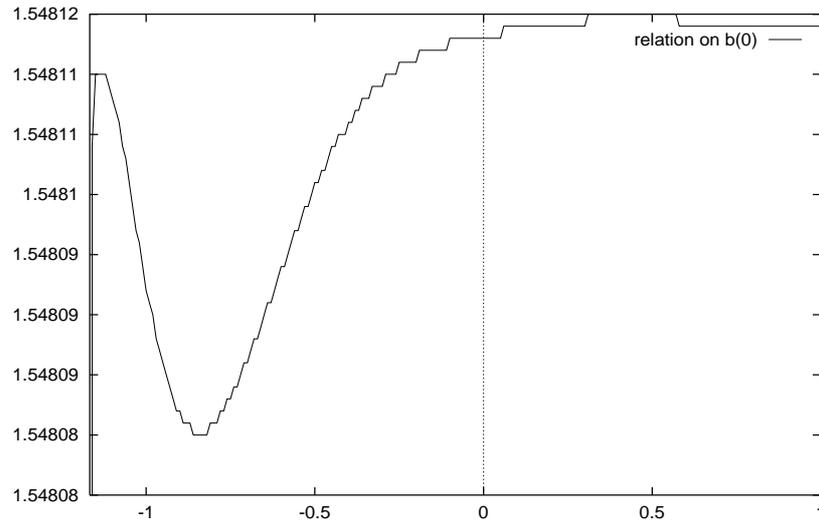


Figure 6.18 Log-sum (6.11) as a function of the parameter $b_0 \in [-1.167, 1]$ regarding the $\$/\pounds$ exchange rate

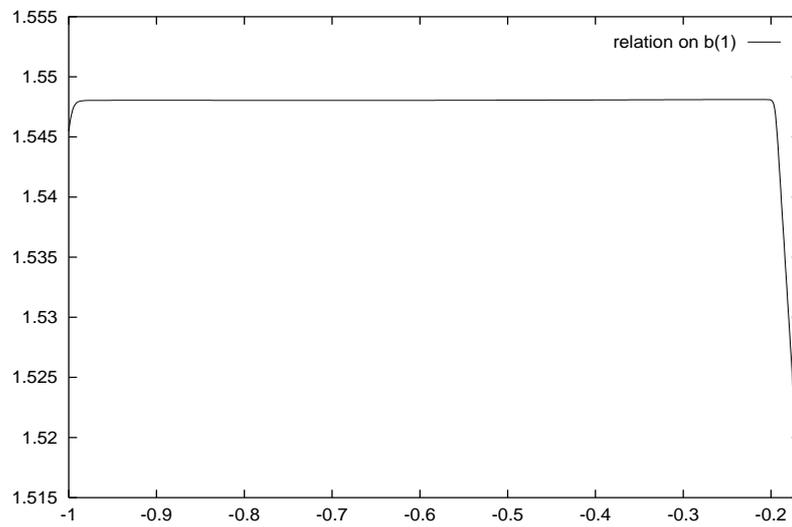


Figure 6.19 Log-sum (6.11) as a function of the parameter $b_1 \in [-1.0, -0.167]$ regarding the $\$/\pounds$ exchange rate

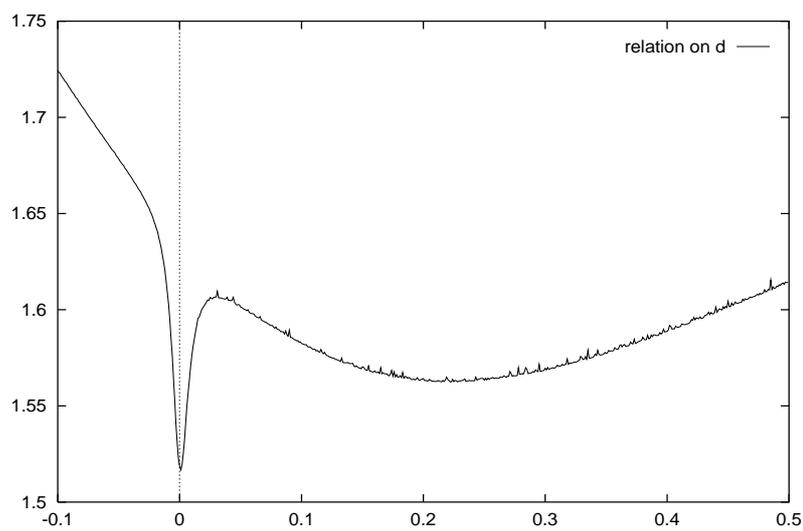


Figure 6.20 Log-sum (6.11) as a function of the parameter $d \in [-0.1, 0.5]$ regarding the $\$/\pounds$ exchange rate

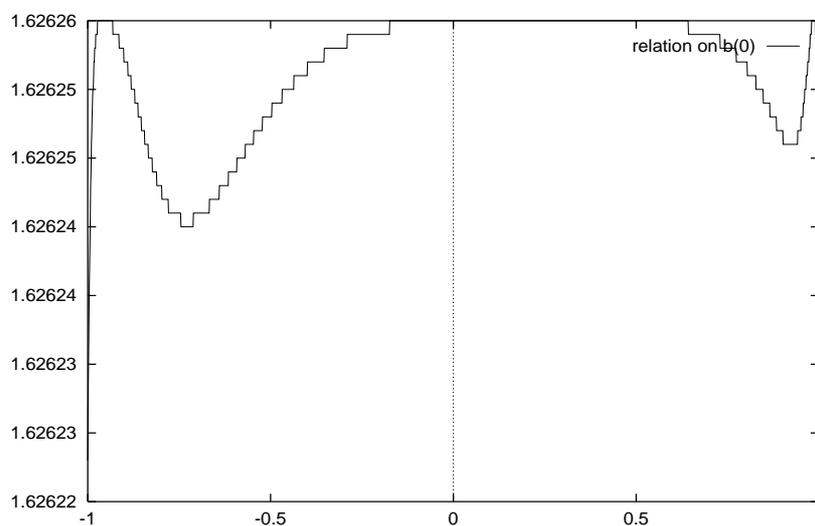


Figure 6.21 Log-sum (6.11) as a function of the parameter $b_0 \in [-1.1, 1.1]$ regarding the DM/ $\$$ exchange rate

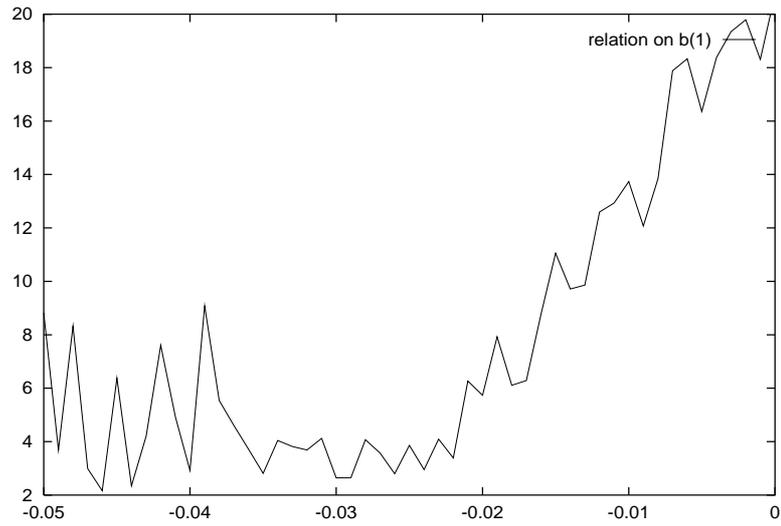


Figure 6.22 Log-sum (6.11) as a function of the parameter $b_1 \in [-0.05, 0.]$ regarding the DM/\$ exchange rate

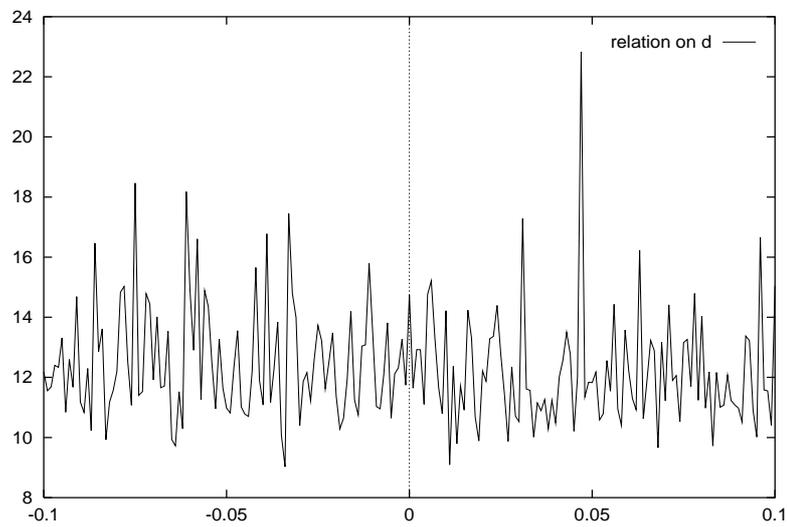


Figure 6.23 Log-sum (6.11) as a function of the parameter $d \in [-0.1, 0.1]$ regarding the DM/\$ exchange rate

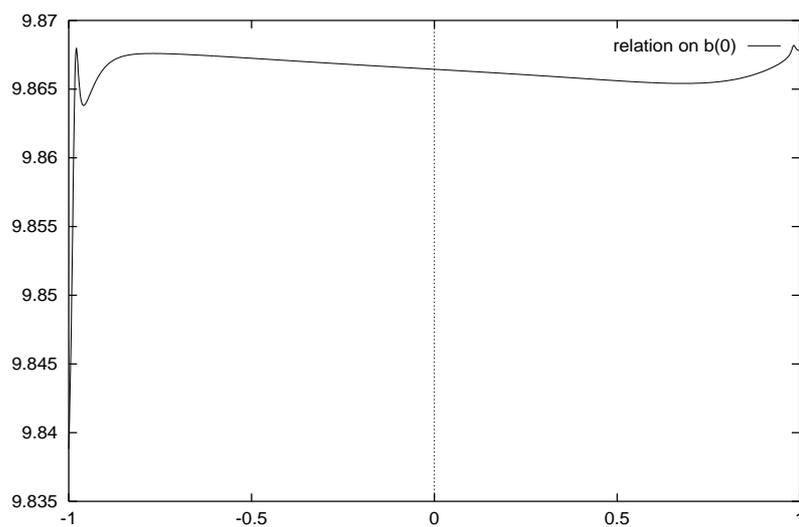


Figure 6.24 Log-sum (6.11) as a function of the parameter $b_0 \in [-1, 1]$ regarding the AT&T stocks closing rate

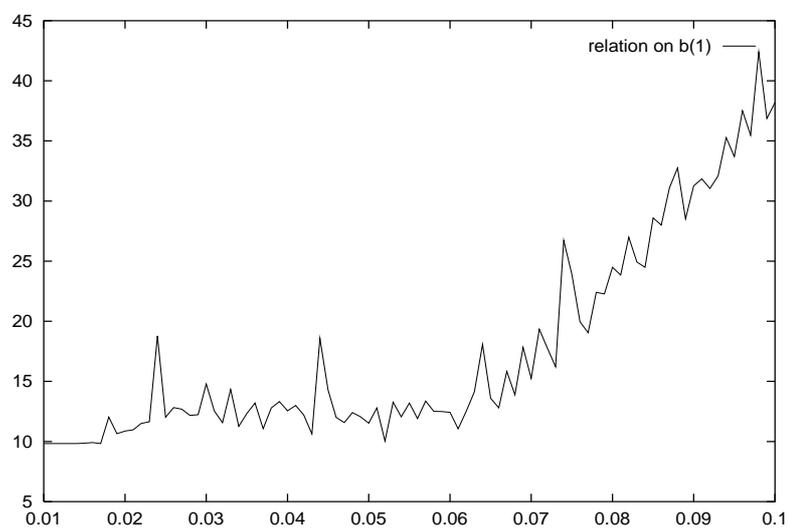


Figure 6.25 Log-sum (6.11) as a function of the parameter $b_1 \in [0.01 : 0.1]$ regarding the AT&T stocks closing rate

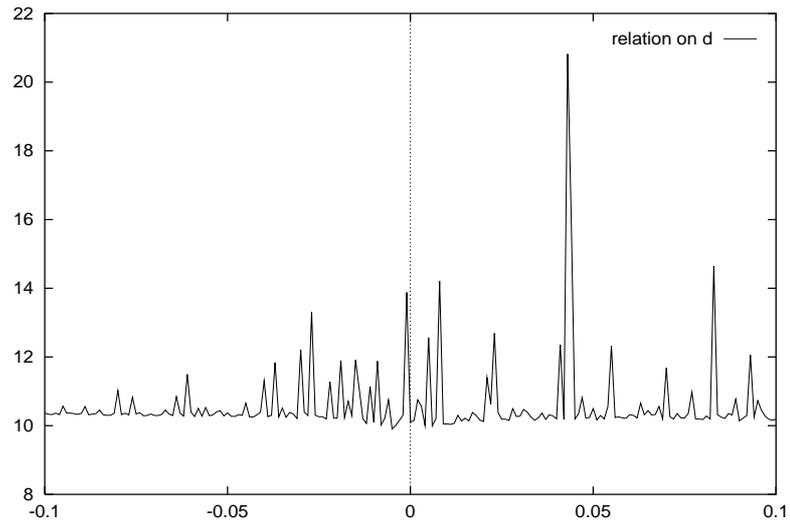


Figure 6.26 Log-sum (6.11) as a function of the parameter $d \in [-0.1, 0.1]$ regarding the AT&T stocks closing rate

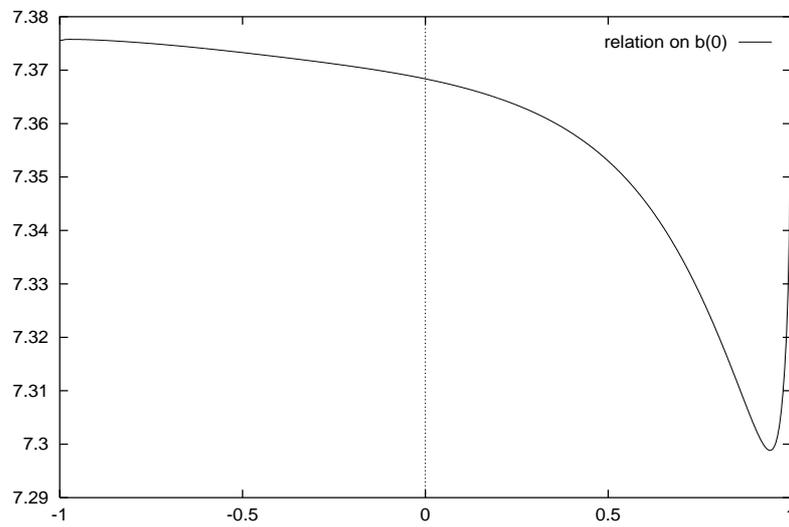


Figure 6.27 Log-sum (6.11) as a function of the parameter $b_0 \in [-1., 1.]$ regarding the Intel Co stocks closing rate

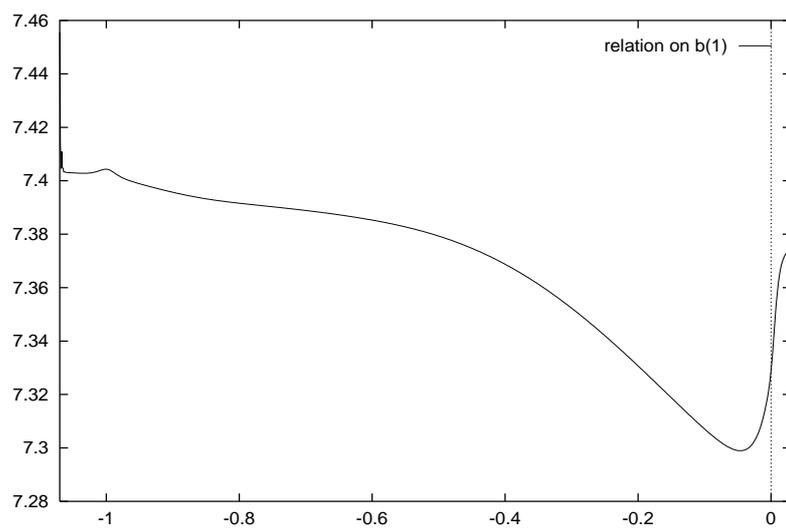


Figure 6.28 Log-sum (6.11) as a function of the parameter $b_1 \in [-1.07, 0.03]$ regarding the Intel Co stocks closing rate

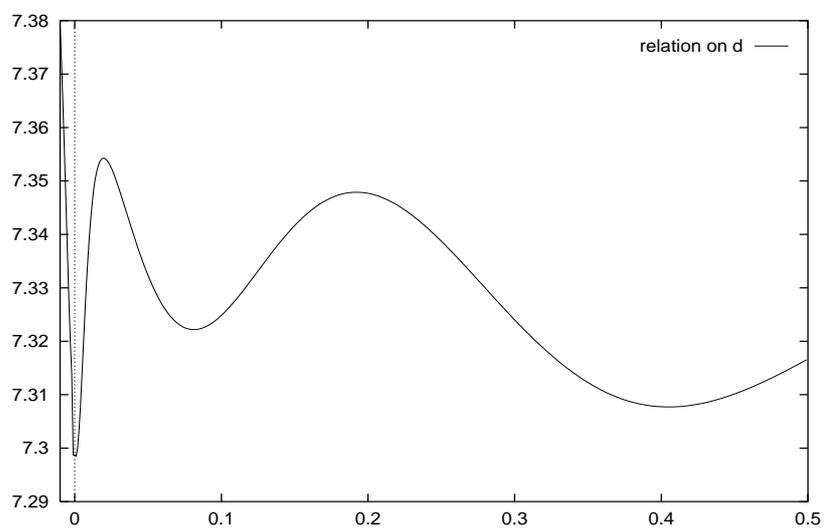


Figure 6.29 Log-sum (6.11) as a function of the parameter $d \in [-0.01, 0.5]$ regarding the Intel Co stocks closing rate

Table 6.1 The initial points obtained in optimizing ARFIMA models by global algorithms

<i>Data</i>	b_0	b_1	d	$\min \log f(x)$
\$/£	-1.195	-0.169	0.0005	1.51675
DM/\$	-1.019	0.0120	0.0007	1.60065
Rial/\$ (1)	0.15	0.45	0.74	0.9395
Rial/\$ (2)	0.51	-0.45	0.0004	0.9350
AT&T	-1.017	0.0118	0.00005	9.83208
Intel Co	0.9975	0.0055	0.012	7.35681

6.6.2 "Initial" Points

The "initial" points b_0^0 , b_1^0 , d^0 (see Table 6.1) for drawing all these relations were defined using a sequence of two global methods referred to as BAYES1 and EXKOR (see [100] and Chapter 19). BAYES1 denotes a search in accordance with a multi-dimensional Bayesian model (4.11). The best result of BAYES1 is a starting point for an one-dimensional coordinate search EXKOR using an one-dimensional Wiener model, see expression (4.4). In both the cases 1000 iterations are used.

The number of auto-regression (AR) parameters is $p = 10$, the number of moving-average (MA) parameters is $q = 2$, and the truncation parameter (6.7) is $R = 30$. The reason is that increasing these numbers we did not succeed to improve the objective function (6.11) significantly.

Table 6.1 shows the estimated parameters for \$/£, and DM/\$ daily exchange rates, and for closing rates of AT&T and Intel Co stocks. Table 6.1 also shows these parameters for rial/\$ monthly exchange rates for two periods: period (1) is before the Iranian revolution, and period (2) is after it. It is clear from Table 6.1 that d 's for all the time series (except the rial/\$ ones) are very close to zero. Hence, we may conclude that the underlying stochastic processes generating \$/£ and DM/\$ rates do not exhibit persistence.

Let us to compare this result with that of traditional approaches to testing for long memory processes (see, for example, [16, 161, 17]). The traditional methods are based on the assumption of continuity: small changes in data do not cause jumps in parameter estimates. This assumption is valid in a linear regression. However, in a non-linear regression, as is the case here, the multi-

modality cannot be ignored because a sum of non-convex functions (6.11) could be multi-modal.

In multi-modal cases, a small change in data may result in a jump of the optimal parameter value. For example, a small change in data may cause the jump of optimal parameter d from zero. Say we obtained optimal $d = 0$ considering the rial/\$ post-revolution data (see Fig. 6.7). We obtain optimal $d = 0.74$ using the pre-revolution data (see.6.11). Investigating various figures we find that slight data changes cause jumps in parameters b_0 and b_1 , too.

The present research clearly shows that the traditional uni-modality assumption used in the linear regression models, is not sustainable for the data sets used in this study and which show evidence of non-linearity. Ignoring the multi-modality of (6.11) one cannot obtain reliable estimates of parameters b and d .

The objective of this work is merely to show a multi-modality of the problem. Therefore to save the computing time the global optimization was carried out approximately using not many of iterations. The results of global optimization were used as a starting point for local optimization. Thus we guarantee the final results at least as good as that of local optimization.

The high-accuracy global optimization is very expensive. As usual, the computing time is an exponential function of accuracy in the global optimization. Therefore, what happens after the high-accuracy global optimization of the objective function, is not yet clear. However, it seems clear that the investigation of multi-modality should be the first step in estimating parameters of non-linear regression models, including the ARFIMA ones⁹. Balancing computing expenses and accuracy of estimation is the important problem of future investigation in the fields of exchange rate prediction and global optimization.

6.7 OPTIMIZATION OF ARMA MODELS

In the previous sections we considered, in general terms, the optimization of parameters of ARFIMA models. Here we consider an algorithm for optimization of parameters of the ARMA model. This model is much simpler and may be regarded as a good first approximation. Denote by y_t the value of y at

⁹Meaning that the sum (6.11) of the ARFIMA model is a non-linear function of the parameters b .

the moment t . Denote by $a = (a_1, \dots, a_p)$ a vector of auto-regression (AR) parameters, and by $b = (b_1, \dots, b_q)$ a vector of moving-average (MA) parameters.

$$y_t - \sum_{i=1}^p a_i y_{t-i} = \epsilon_t - \sum_{j=1}^q b_j \epsilon_{t-j}, \quad t = 1, \dots, T. \quad (6.38)$$

The residual

$$\epsilon_t = y_t - \sum_{i=1}^p a_i y_{t-i} + \sum_{j=1}^q b_j \epsilon_{t-j} \quad (6.39)$$

or

$$\epsilon_t = B_t + \sum_{i=1}^p a_i A_t(i). \quad (6.40)$$

Here

$$B_t = y_t + \sum_{j=1}^q b_j B_{t-j-1} \quad (6.41)$$

and

$$A_t(i) = -y_{t-i-1} + \sum_{j=1}^q b_j A_{t-j-1} \quad (6.42)$$

where $t - i > 0$ and $t - j > 0$.

6.7.1 Optimization of AR parameters

Denote

$$S(a, b) = \sum_{t=1}^T \epsilon_t^2, \quad (6.43)$$

where $a = (a_1, \dots, a_p)$ and $b = (b_1, \dots, b_q)$.

From expressions (6.43) and (6.40) the minimum condition is

$$\frac{\partial S(a, b)}{\partial a_j} = 2 \sum_{t=1}^T \epsilon_t A_t(j) = 0, \quad j = 1, \dots, p \quad (6.44)$$

or

$$\sum_{i=1}^p A(i, j)a_i = -B(j), \quad j = 1, \dots, p, \quad (6.45)$$

where

$$A(i, j) = \sum_{t=1}^T A_t(i)A_t(j) \quad (6.46)$$

and

$$B(j) = \sum_{t=1}^T A_t(j)B_t. \quad (6.47)$$

The minimum of expression (6.43) at fixed parameters b is defined by a system of linear equations:

$$a(b) = A^{-1}B. \quad (6.48)$$

Here matrix $A = (A(i, j), i, j = 1, \dots, p)$ and vector $B = (B(j), j = 1, \dots, p)$, where elements $A(i, j)$ are from (6.46), components $B(j)$ are from (6.47), and A^{-1} is an inverse matrix A . This way one define the vector $a(b) = (a_i(b), i = 1, \dots, p)$ that minimize sum (6.43) at fixed parameters b .

6.7.2 Optimization of MA parameters

The sum of squared residuals (6.43) is a nonlinear non-convex function of MA parameters b . Thus we have to consider the global optimization algorithms. Denote

$$f(x) = \log S(a(x), x), \quad (6.49)$$

where $x = b$ and $S(a, b)$ is from (6.43) at optimal parameter $a = a(b)$. Denote

$$b^0 = x^0 = \arg \min_x f(x). \quad (6.50)$$

6.7.3 Predicting "Next-day" Rate

We minimize the expected "next-day" squared deviation ϵ_{t+1} using the data available at the moment t

$$y_{t+1}^0 = \arg \min_{y_{t+1}} \mathbf{E} \epsilon_{t+1}^2. \quad (6.51)$$

Here

$$\mathbf{E} \epsilon_{t+1}^2 = \mathbf{E} (B_{t+1} + \sum_{i=1}^p A_{t+1}(i) a_i(b^0))^2, \quad (6.52)$$

where the optimal parameter b^0 was obtained using the data available at the day t . Variance (6.52) is minimal, if

$$y_{t+1}^0 = B_{t+1} + \sum_{i=1}^p A_{t+1}(i) a_i(b^0), \quad (6.53)$$

because the expectation of y_{t+1} is $B_{t+1} + \sum_{i=1}^p A_{t+1}(i) a_i(b^0)$ under the assumptions.

6.7.4 "Continuous" model

If we wish to keep the "continuity" of sample functions as time unit tends to zero, we have to consider a special case when $a_1 = 1$. In such a case, one ought to change expressions (6.38)-(6.53), correspondingly. The popular Random Walk (RW) model:

$$y_{t+1} = y_t + \epsilon_t, \quad (6.54)$$

may be regarded as a special case of the "continuous" model when $q = 0$.

6.7.5 Examples

We compare the "next-day" ARMA prediction results and a popular Random Walk (RW) model, where the conditional expectation of $y_{t+1} = y_t$. Table 6.2 shows the difference between the mean square deviations of ARMA and RW models using DM/\$ and \$/£ exchange rates and AT&T and Intel Co. stocks

Table 6.2 The average prediction results of ARMA and RW models

<i>Data</i>	<i>ARMA</i>	<i>RW</i>	<i>ARMA - RW</i>
\$/£	2.299928e-02	2.262379e-02	3.754923e-04
DM/\$	3.912910e-02	3.968943e-02	-5.603285e-04
AT&T	1.677580e+02	1.644688e+02	3.289204e+00
Intel Co	4.063805e+02	4.133900e+02	-7.009471e+00

closing rates for the period of $T = 460$ days. Let us denote by $T_0 = T/4 = 115$ the number of days, used for the "initial" parameter estimation employing global optimization methods. This number seems sufficient for the initial estimation. The "initial" estimates are updated later on by local methods. *ARMA* and *RW* denote the mean square deviations of ARMA and RW "next-day" predictions. The difference is denoted as *ARMA - RW*.

Table 6.2 shows the average over 345 "next-day" predictions. The table demonstrates that the ARMA model predicts the DM/\$ exchange rate and the Intel Co. closing rate better than RW. For the \$/£ exchange rate and the AT&T closing rate the opposite is true. The difference is slight but not so insignificant since the average of 345 "next-day" predictions is shown.

There are formal significance tests to answer to this type of questions. However, the results depend on the estimate distribution which is not well-defined in multi-modal cases. The reason is the discontinuity of multi-modal estimates since even slight data changes my cause jumps of estimates.

The results of traditional significance tests depend on the observation numbers, too. For instance, it is shown (see [98]) that any positive difference will become "significant", if the observation number is sufficiently large. Therefore, let us merely define the average prediction errors (see Table 6.2) and declare that the number of observations is 345. Table 6.3 shows optimal parameters $a(b) = (a_0(b), \dots, a_9(b))$ and $b = (b_0, b_1)$ of the ARMA model used for the first 10 of 345 "next-day" predictions of the DM/\$ exchange rate. The values of the objective function $f(x)$, $x = (b_0, b_1)$ are denoted by v .

Finishing this chapter it seems appropriate to cite a remark made by our colleague Professor of Economics Abdol Soofi made after reading the manuscript:

Table 6.3 The optimal parameters a and b of ARMA model for 10 predictions

a=	1.6261e+00	-1.5934e+00	9.650e-01	0.000e+00	-1.999e+00
	0.0000e+00	-1.9986e+00	0.0000e+00	0.0000e+00	0.0000e+00
b=	6.2827e-01	-9.7024e-01	v=	-4.9132e+00	
a=	1.6126e+00	-1.5691e+00	9.5426e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	6.0840e-05	6.2966e-05	-7.8000e-03
b=	6.1477e-01	-9.5945e-01	v=	-4.9053e+00	
a=	1.6085e+00	-1.5632e+00	9.5250e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	7.6840e-05	7.1677e-05	-4.0000e-03
b=	6.1063e-01	-9.5766e-01	v=	-4.9041e+00	
a=	1.6197e+00	-1.5866e+00	9.6464e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	4.8488e-04	4.2047e-04	-2.0200e-02
b=	6.2186e-01	-9.6989e-01	v=	-4.8595e+00	
a=	1.6284e+00	-1.6028e+00	9.7224e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	5.9513e-04	4.9755e-04	-1.0500e-02
b=	6.3051e-01	-9.7745e-01	v=	-4.8502e+00	
a=	1.6322e+00	-1.6079e+00	9.7354e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	6.5913e-04	5.5094e-04	-8.0000e-03
b=	6.3427e-01	-9.7880e-01	v=	-4.8437e+00	
a=	1.6388e+00	-1.6254e+00	9.8438e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	8.2813e-04	7.1553e-04	1.3000e-02
b=	6.4082e-01	-9.8975e-01	v=	-4.8243e+00	
a=	1.6348e+00	-1.6177e+00	9.8062e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	8.5517e-04	7.4322e-04	-5.2000e-03
b=	6.3693e-01	-9.8600e-01	v=	-4.8212e+00	
a=	1.6351e+00	-1.6236e+00	9.8619e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	1.0597e-03	8.8185e-04	-1.4300e-02
b=	6.3722e-01	-9.9145e-01	v=	-4.8049e+00	
a=	1.6382e+00	-1.6311e+00	9.9063e-01	0.0000e+00	-1.9991e+00
	0.0000e+00	-1.9986e+00	1.1319e-03	9.0880e-04	-8.5000e-03
b=	6.4026e-01	-9.9593e-01	v=	-4.8020e+00	

There is a lingering debate between the academicians who believe that financial markets behave randomly and the chartists, the practitioners, who believe that there are profits to be made by manipulating financial data regardless whether the markets behave randomly. While the academicians press on with their theoretical constructs in supporting the random walk models, the practitioners make fortunes in the assets markets, and in the process increase income disparity between themselves and the academicians.

Note that The Wall Street Journal does an experiment regarding the comparative advantages of "experts" vs. the RW. The experiment goes something like this. They get a number of experts, money managers, to recommend a number of stocks. Then they select an equal number of stocks by throwing darts on a page of the Wall Street Journal with a listing of stocks. Then after a period of time, they compare the returns on both set of stocks in order to determine which group has the highest return. Which group has won? Well, it usually varies. No definitive results thus far.

The ARMA optimization is doing about the same. We may improve the performance of time series by non-linear models and by more exact optimization. The Wall Street Journal may improve the experts results, too.

7

OPTIMIZATION PROBLEMS IN SIMPLE COMPETITIVE MODEL

7.1 INTRODUCTION

We consider here optimization problems of a simple competitive model. There are several servers providing the same service. Each server fixes the price and the rate of service. The rate of service defines the customers time losses while waiting in queue for the service. The customer goes to the server with lesser total service cost. The total cost includes the service price plus waiting losses. The customer goes away, if the total cost exceeds a certain critical level. Both the flow of customers and the service time are stochastic. There is no known analytical solution for this model. The results are obtained by Monte-Carlo simulation. The analytical solution of a simplified model is considered.

The model is used to illustrate the possibilities and limitations of the optimization theory and numerical techniques in competitive models. We consider optimization in two different mathematical frameworks: the fixed point and Lagrange multipliers. Two different economic and social objectives are considered: the equilibrium and the social cost minimization.

The competitive model is applied as a test function for the Bayesian algorithms. However, a simple model may help to design more realistic ones describing the processes of competition better. Besides, one may use the competitive model for teaching Operations Research, too.

7.2 COMPETITIVE MODEL

Let us consider m servers providing the same service:

$$u_i = u_i(x_1, x_2, y_1, y_2) = a_i y_i - x_i, \quad i = 1, \dots, m, \quad (7.1)$$

where u_i is the profit, y_i is the service price, x_i is the running cost, a_i is the rate of customers, and i is the server index. Assume, for simplicity, that a service rate¹ is equal to the running cost x_i . The service cost

$$c_i = y_i + \gamma_i, \quad (7.2)$$

where γ_i is waiting cost. Assume that the waiting cost is equal to an average waiting time at the server i . A customer goes to the server i , if

$$c_i < c_j, \quad j = 1, \dots, m, \quad j \neq i, \quad c_i \leq c_0. \quad (7.3)$$

A customer goes away, if

$$\min_i c_i > c_0, \quad (7.4)$$

where c_0 is the critical cost. The rate a of incoming consumers flow is fixed:

$$a = \sum_{i=0}^m a_i, \quad (7.5)$$

where a_0 is the rate of lost customers.

Conditions (7.3) and (7.4) separate the flow of incoming customers into $m + 1$ flows thus making the problem very difficult for analytical solution. The separated flow is not simple even in the Poisson incoming flow case [52]. Thus we need Monte Carlo simulation, to define the average rates of customers a_i , $i = 0, 1, \dots, m$, by conditions (7.3) (7.4), and the average profits u_i , $i = 1, \dots, m$, by expression (7.1).

7.3 SEARCH FOR EQUILIBRIUM

We fix the "initial contract", the initial values x_i^0, y_i^0 , $i = 1, \dots, m$. The "new contract" x_i^1, y_i^1 , $i = 1, \dots, m$, is obtained by maximizing the profits of each

¹Average number of customers per unit time

server i , under the assumption that all the partners $j \neq i$ will honor the initial contract x_i^0, y_i^0 , $i = 1, \dots, m$

$$(x_i^1, y_i^1) = \arg \max_{x_i, y_i} u_i(x_i, y_i, x_j^0, y_j^0, j = 1, \dots, m, j \neq i), \quad i = 1, \dots, m. \quad (7.6)$$

Condition (7.6) transforms the vector z^n , $n = 0, 1, 2, \dots$ into the vector z^{n+1} . Denote this transformation by T

$$z^{n+1} = T(z^n), \quad n = 0, 1, 2, \dots \quad (7.7)$$

Here the vector $z = (x_i, y_i, i = 1, \dots, m) \in B \subset R^{2m}$. We obtain the equilibrium at the fixed point z^n , where

$$z^n = T(z^n). \quad (7.8)$$

The fixed point z^n exists, if the feasible set B and the profit functions (7.1) are all convex [90]. We obtain the equilibrium directly by iterations (7.7), if the transformation T is contracting [109]. If not, then we minimize the square deviation

$$\min_{z \in B} \|z - T(z)\|^2. \quad (7.9)$$

The equilibrium is achieved, if the minimum (7.9) is zero. If the minimum (7.9) is positive then the equilibrium does not exist. One minimize (7.9) by the usual stochastic approximation techniques [39]), if square deviation (7.9) is unimodal. If not, then the Bayesian techniques of global stochastic optimization (see [100] and Chapter 4) are used.

Obviously an equilibrium will be stable if transformation (7.7) is locally contracting in the vicinity of fixed point (7.8). If not, then some stabilizing conditions should be introduced. The transformation $T(z)$ is referred to as locally contracting if there exists a constant $0 \leq \alpha < 1$ such that

$$\|T(z^1) - T(z^2)\| \leq \alpha \|z^1 - z^2\| \quad (7.10)$$

for all $z^1, z^2 \in Z_\epsilon$, where Z_ϵ is an ϵ -vicinity of fixed point defined by the "natural" deviations from the equilibrium.

7.3.1 Simplified Illustration

To illustrate the idea of equilibrium we consider very simple deterministic model. We express the waiting time as

$$\gamma_i = a_i/x_i, \quad i = 1, 2. \quad (7.11)$$

We may compare the simplified expression (7.11) with the well-known expression of average waiting time in the Poisson case, see [52]

$$\gamma_i = \frac{a_i}{x_i} \frac{1}{x_i - a_i}. \quad (7.12)$$

Assume the steady-state conditions²

$$a_i/x_i + y_i = q. \quad (7.13)$$

Here q is a steady-state factor. From expression (7.4)

$$q \leq c. \quad (7.14)$$

From steady-state conditions (7.13)

²This example is merely an illustration, stability of equilibrium conditions is not considered here.

$$a_i = (q - y_i)x_i \quad (7.15)$$

and

$$u_i = (q - y_i)x_i y_i - x_i \quad (7.16)$$

Maximizing the profit

$$\begin{aligned} \max_{x_i, y_i, q} x_i((q - y_i)y_i - 1) \\ 0 \leq x_i \leq a, \quad q \leq c \end{aligned} \quad (7.17)$$

we obtain the optimal values

$$x_i = a, \quad y_i = q/2, \quad i = 1, 2, \quad q = c. \quad (7.18)$$

From expressions (7.17) and (7.18) the maximal profit

$$u_i = a((c/2)^2 - 1) \quad (7.19)$$

We achieve a positive profit equilibrium, if $c > 2$. These results may be helpful understanding the model (7.1).

7.3.2 Monte-Carlo Simulation

Assume that the n -th customer estimates the average waiting time at the server i as the relation

$$\gamma_i(n) = n_i/n, \quad n = 1, \dots, N \quad i = 1, \dots, m, \quad (7.20)$$

where n_i is queue length at the server i when the n -th customer arrives. Then from expressions (7.2) and (7.20) the service cost

$$c_i = y_i + n_i/n, \quad n = 1, \dots, N \quad i = 1, \dots, m. \quad (7.21)$$

Alg.	Prices y_i and Rates x_i						Profits u_i			Object.
No.	x_1	y_1	x_2	y_2	x_3	y_3	u_1	u_2	u_3	$\min f$
1	0.04	0.95	2.45	8.30	1.31	1.25	0.03	2.87	1.50	1.78
2	3.24	8.81	0.13	0.09	0.45	1.32	9.48	0.02	0.77	0.45
3	0.63	6.87	0.63	1.88	3.13	9.38	2.73	2.15	4.67	0.88
4	0.01	0.95	2.45	8.30	1.31	1.25	0.01	3.46	1.50	1.48
5	2.23	3.56	0.73	0.38	3.52	2.99	0.12	0.09	0.13	1.72

Algorithm Numbers				
1	2	3	4	5
MIG1	BAYES1	LPMIN	EXKOR	GLOPT

Table 7.1 Simulation Results

Table 7.1 indicates the possibilities and limitations of direct Monte-Carlo simulation of transformation (7.7) using different algorithms of global optimization.

The simulation parameters are:

$m = 3$, $N = 500$, $a = 2$, $c = 12$.

The constraints are:

$0.0001 \leq x_i \leq 10$, $0.0001 \leq y_i \leq 10$, $i = 1, 2$.

The positive result is that we obtained relatively small deviation from the equilibrium (0.45 using BAYES1 and 0.88 using LPMIN). However, we need much greater accuracy to answer a number of questions, for example:

- why using algorithm 2 we obtained the profit u_1 which is much greater as compared with u_2 and u_3 in the symmetric conditions?
- why using different algorithms we obtained so different results?

- is the equilibrium solution unique?
- is the algorithm accuracy sufficient?

The visual inspection shows that the profit functions u_i look like convex. That indicates the existence of the equilibrium. However one needs a large amount of computing to obtain the answers to these and related questions. That is outside the objective of this book because we consider the competitive and the other models mainly as test functions to compare different algorithms including the Bayesian ones. Table 7.1 shows that in this special case the best results we obtained by the Bayesian methods and the second best by the uniform search algorithms during comparable time (see lines 2 and 3 correspondingly).

7.4 "SOCIAL" MODEL

Define the "social" cost of service as follows

$$\sum_i (x_i + a_i \gamma_i). \quad (7.22)$$

Expression (7.22) defines a sum of running and waiting costs. For example, both the running and waiting costs may be defined as a time lost by the members of society while running the servers and waiting in the queues. The prices y are not present in social cost expression (7.22), since the service rates x_i are not limited.

In the simplified case (7.11), the optimal service rates

$$x_i = a_i, \quad i = 1, 2. \quad (7.23)$$

Here the prices y are eliminated, since the of service rates are not limited.

7.5 LAGRANGE MULTIPLIERS

Consider now the Lagrangian model. We limit the total service rate of both servers by b

$$\sum_i x_i \leq b, \quad (7.24)$$

fix the customer rates $a_i, i = 1, 2$, and minimize the service losses

$$\min_x \sum_i a_i \gamma_i. \quad (7.25)$$

Problem (7.25) (7.24) can be solved by Lagrange multipliers, assuming the convexity of γ_i

$$\max_{y \geq 0} \min_{x \geq 0} \left(\sum_i a_i \gamma_i + y \left(\sum_i x_i - b \right) \right). \quad (7.26)$$

Here the Lagrange multiplier y means the "price" which the server pays to the supplier of service resources x_i . First let us fix y and minimize (7.26) by x . Thus the optimal rate $x_i = x_i(y)$ is defined as a function of price y . Next we maximize (7.26) by $y \geq 0$ to obtain the equilibrium (max-min) price $y = y_0$. Now each server can define the optimal service rate $x_i = x_i(y_0)$ by minimizing the social service cost

$$\min_{x_i \geq 0} (y_0 x_i + a_i \gamma_i), \quad i = 1, 2. \quad (7.27)$$

Apparently this model is not quite competitive, since the customer rate is fixed for each server. One defines equilibrium between the supplier and the servers. Here we assume a competition not between servers, like in (7.1), but between the supplier and the servers. The servers are of "non-profit" type. They minimize the social service cost including the customer waiting losses γ_i plus the price $y_0 x_i$ paid by the server to obtain the resource x_i .

In the simplified case (7.11),

$$\max_{y \geq 0} \min_{x \geq 0} \left(\sum_i a_i^2 / x_i + y \left(\sum_i x_i - b \right) \right). \quad (7.28)$$

First we fix y and obtain optimal $x_i = x_i(y)$ as a function of y

$$x_i(y) = a_i / \sqrt{y}. \quad (7.29)$$

Next we maximize by y

$$\begin{aligned} \max_{y \geq 0} (\sqrt{y} \sum_i a_i + y(1/\sqrt{y} \sum_i a_i - b)) = \\ \max_{y \geq 0} (2a\sqrt{y} - by) \end{aligned} \quad (7.30)$$

and obtain the optimal price

$$y = y_0 = (a/b)^2. \quad (7.31)$$

This and expression (7.29) imply

$$x_i = a_i/ab. \quad (7.32)$$

All the solutions of simplified models are illustrative. However, they may be used as a first approximation considering more complicated models, correctly representing the stochastic service and processes.

PART III

NETWORKS OPTIMIZATION

8

APPLICATION OF GLOBAL LINE-SEARCH IN THE OPTIMIZATION OF NETWORKS

8.1 INTRODUCTION

An application of Global Line-Search (GLS) to the optimization of networks is considered in this chapter. Advantages and disadvantages are discussed. It is shown that GLS provides the global minimum after a finite number of steps in two cases of piecewise linear cost functions of arcs. The first case is, where all cost functions are convex. The second case is, where all costs are equal to zero at zero flow, and equal to some constant at non-zero flow. In other cases the global line-search approaches the global minimum with a small average error. Therefore this algorithm is regarded as a good heuristics.

An extension of the method to vector demands is given. The application of the method to the optimization of the high-voltage power system networks is described.

8.2 GLOBALLINE-SEARCH (GLS)

Suppose that the objective function $f(x)$, $x = (x_j, j = 1, \dots, J)$ may be approximately expressed as a sum of components depending on one variable x_j .

$$f(x) = \sum_{j=1}^J f_j(x_j). \quad (8.1)$$

Then the original J -dimensional optimization problem can be reduced to a sequence of one-dimensional optimization problems. If decomposition (8.1) is

exact, then we obtain the global optimum after J steps of optimization. If sum (8.1) represents $f(x)$ approximately, then we obtain some approximation of global optimum.

The result of step i is regarded as the initial point for the $i + 1$ -th step of optimization. The optimization stops, if no change occurs during J steps. The difference from the classical version of line-search method is that the search is not local but global. As usual, it helps to approach the global minimum closer. There are important cases when the global line-search reaches the global minimum, and the local line-search does not. One of such cases is the following problem of network optimization.

8.3 OPTIMIZATION OF NETWORKS

Suppose that the cost of network $f(x)$ can be expressed as sum (8.1), where $f_j(x_j)$ is the cost of arc j and x_j is the flow of arc j . The sum of flows of arcs connected to each node i has to be equal to the node demand c_i . It means that:

$$\sum_{j=1}^J a_{ij}x_j = a_i c_i, \quad i = 1, \dots, I \quad (8.2)$$

where

$$\sum_{i=1}^{i=I} a_i c_i = 0. \quad (8.3)$$

Here J is the number of arcs, I is the number of nodes, c_i is a demand of the node i ,

$$a_{ij} = \begin{cases} +1, & \text{if arc } j \text{ goes to node } i \\ -1, & \text{if arc } j \text{ goes from node } i \\ 0, & \text{if arc } j \text{ is not connected to } i \end{cases}$$

and

$$a_i = \begin{cases} +1, & \text{if node } i \text{ is the source} \\ -1, & \text{if node } i \text{ is the sink.} \end{cases}$$

The J -dimensional problem of network cost minimization (8.1) under $I - 1$ of conservation-of-flow equations (8.2) can be reduced to a $J - I + 1$ -dimensional unconstrained minimization problem:

$$f(x) = \sum_{k=1}^K f_k(x_k) + \sum_{l=K+1}^J f_l\left(\sum_{k=1}^K b_{lk}x_k + x_{l0}\right). \quad (8.4)$$

Here $K = J - I + 1$ is the number of loops, x_{l0} is defined by expression (8.2) taking $x_k = 0, k = 1, \dots, K$,

$$b_{lk} = \begin{cases} +1, & \text{if directions of arc } l \text{ and loop } k \text{ are the same} \\ -1, & \text{if directions of arc } l \text{ and loop } k \text{ are opposite} \\ 0, & \text{if arc } l \text{ does not belong to loop } k. \end{cases}$$

Loop k is generated connecting a pair of nodes of some tree containing arcs $l, l = K + 1, \dots, J$ by an additional arc $k, k = 1, \dots, K$. The flow x_k of arc k generating a loop k is called a loop flow. Loop flows $x_k, k = 1, \dots, K$ can be changed independently during optimization.

Expression (8.4) depends on the tree which generates loops $k, k = 1, \dots, K$. Assume that the costs of arcs are piecewise linear functions. Then, it is convenient to carry out the global line-search along the bounds of linear areas. It can be done by changing the tree after each step of optimization. An obvious rule is to remove some arc from the tree, if the arc flow happens to be on the bound of linear parts of a piece-wise linear cost function. We do not consider degenerate problems. It is shown (see [98]), that this algorithm provides global minimum in two cases:

- cost functions of all arcs are convex;
- cost functions of all arcs are constant, with an exception of the zero flow point. At zero flow, the value of the cost function has to be zero. It means that cost functions are "very" non-convex.

It is easy to see that in the linear case the algorithm is as simple and efficient as conventional algorithms of linear programming. The difference is that the global line-search algorithm works almost as well in non-linear convex problems and in some special non-convex problems, too.

A generalization of the algorithm to S -dimensional demand is straightforward theoretically: we just replace a scalar demand c_i by a vector demand

$c_i = (c_{i1}, \dots, c_{iS})$. To keep conservation-of-flow equations (8.2) we have to replace scalar flows x_j by the corresponding vector flows $x_j, x_j = (x_{j1}, \dots, x_{jS})$. In practical applications a straightforward generalization is not convenient for two reasons:

- an exponential growth of calculations when S is large;
- practical difficulties defining general S -dimensional piecewise linear functions $f_j(x_{j1}, \dots, x_{jS})$.

In special S -dimensional demand cases, the global line-search can be carried out more conveniently using specific heuristics. The proof of convergence can be extended directly only for a straightforward generalization. However, the experience shows that, as usual, the global line-search is efficient in solving S -dimensional load problems of network design. One of such problems is the optimization of a high-voltage net of a large power system.

8.4 OPTIMIZATION OF HIGH-VOLTAGE POWER SYSTEM NETWORKS

Arcs represent components of a network such as power transmission lines and transformers. Nodes represent demands. Demands are sources (generators or higher voltage sub-stations) or sinks (users or lower voltage sub-stations). The demands of nodes $c_i = (c_{in}, n = 1, \dots, N)$ in real life power systems are some vector-valued functions of time $c_{in} = c_{in(t)}, t \in [1, T]$. Usually these functions are approximated as some step functions of time, so

$$c_i = (c_{int}, n = 1, \dots, N, t = 1, \dots, T), i = 1, \dots, I.$$

Here different t -components of $S = NT$ -dimensional demand represent different periods of time, from $t = 1$ to $t = T$. Then flows of arcs are $x_j = (x_{jnt}, n = 1, \dots, N, t = 1, \dots, T), j = 1, \dots, J$ and scalar problem (8.4) can be directly extended to the vector case:

$$f(x) = \sum_{k=1}^K f_k(x_k) + \sum_{l=K+1}^J f_l\left(\sum_{k=1}^K b_{lk} x_k + x_{l0}\right), \quad (8.5)$$

where

$$x_k = (x_{knt}, n = 1, \dots, N, t = 1, \dots, T), k = 1, \dots, K \quad (8.6)$$

The costs of arcs j representing transmission lines and transformers j directly depend not only on flows x_j but also on states y_j . So the arc cost can be more conveniently expressed as $f_j(x_j, y_j)$. Here the state variable $y_j = (y_{j,t}, t = 1, \dots, T)$ usually depends on time but not on n .

Each component y_{jt} of the vector y_j is a non-negative integer defining technical parameters of arc j , such as the number of parallel circuits of the transmission line, the number and cross-section area of wires, the number and power rating of transformers, and so on. Assume that the capacity of arc $X_{jt}(y_{jt})$ is an increasing function of its state y_{jt} (this assumption will help us later to deal with capacity constraints). So we define a mixed integer programming problem:

$$f(x, y) = \sum_{k=1}^K f_k(x_k, y_k) + \sum_{l=K+1}^J f_l(\sum_{k=1}^K b_{lk} x_k + x_{l0}, y_l). \quad (8.7)$$

This problem can be reduced to a continuous non-linear programming problem by choosing the cheapest state y_j for a fixed flow x_j , namely:

$$f_j(x_j) = \min_{y_j \in Y_j} f(x_j, y_j), \quad (8.8)$$

providing that the capacity constraints hold

$$|x_{jnt}| \leq X_{jnt}(y_{jt}) \quad (8.9)$$

The capacity constraints $|x_{jnt}| \leq X_{jnt}(y_{jt})$ are satisfied by increasing the state variable y_{jt} , if inequality (8.9) does not hold. Notation $Y_j = Y_j(x_j)$ means a set of feasible states of arc j that can depend on the flow x_j . So expression (8.8) defines the cost function, generally a multi-modal one.

Expression (8.8) gives a convenient definition of the S -dimensional cost function $f_j(x_j)$. However, there remains an exponential complexity of minimization of non-convex cost function (8.7) depending on vector variables $x_k, k = 1, \dots, K$. So we shall consider some other ways, too.

An interesting way is to reduce the problem of mixed integer programming (8.7) to a problem of pure integer programming, regarding the states y_j as independent integer variables. Here, for each fixed state $y = y_k$ the optimal value of flow x_k has to be calculated.

An advantage of the pure integer programming approach is that the cost of arc $f_j(x_j, y_j)$ at a fixed state y_j is usually a convex function (compare it with non-convex cost function (8.8) of the non-linear programming approach). It is well known that sum (8.7) of convex functions is a convex function, too.

However we should carry out the optimization of convex function (8.7) many times, for each state y . It is a hard task, if N and T are not small. The practical experience shows that the best computational results can be obtained by facing the mixed integer programming problem (8.7) directly. It means that we optimize states and flows together at each step of global line-search.

8.5 MIXED INTEGER LINE-SEARCH

Let us consider a special case

$$f_j(x_j, y_j) = \sum_{t=1}^T [g_{jt}(y_{jt-1}, y_{jt}) + \sum_{n=1}^N h_{jnt}(y_{jt})x_{jnt}^2]. \quad (8.10)$$

Here the function $g_{jt}(y_{jt-1}, y_{jt})$ defines the cost of reconstruction of node j from state y_{jt-1} to state y_{jt} . The expression $h_{jnt}x_{jnt}^2$ means the power loss of flow x_{jnt} in the arc j at the state y_{jt} . It is well known that minimization of (8.7) under assumption (8.10) defines the natural distribution of power flows in a homogeneous electrical net for any fixed state y . The net is not homogeneous if it contains transmission lines of different voltages or if it includes lines and transformers together.

Suppose that $y_{jnt} = 0, if x_{jnt} = 0$ and that $y_{jnt+1} \geq y_{jnt}$. It means that zero state is feasible only for zero flows, and that the state variable can't be decreased in time. The last assumption is usually true, but not always.

If the state of arc is zero from time 1 to time t , we shall define it as a t -interrupted arc. It is supposed that after time t the state of the t -interrupted arc is non-zero. The optimization of each loop k is carried out in $T + 1$ stages.

At the first stage, we compare all possible cases of T -interruption of arcs belonging to the loop k . For each fixed state we minimize sum (8.7) as a function of flow x_k . We do it by solving linear equations corresponding to the condition of zero first derivatives with regard to $x_{knt}, n = 1, \dots, N, t = 1, \dots, T$. In the most economical state of loop we replace zero state values $y_{jt} = 0$ by unit state values, that is by $y_{jt} = 1$ and accept it as the initial state for the next stage.

At the second stage, we consider all cases of $T - 1$ interruption and so on until the last $T + 1$ -st stage. At $T + 1$ -st stage all non-zero states are considered.

We compare sums of the costs of arcs belonging to the loop k for all stages. The state which corresponds to the minimal cost function is accepted as a result of global search along the "line" k . The enumeration of arcs may be changed after each step of global line-search. The purpose of this change is to keep the "most interrupted" arcs out of the tree.

An arc is called "most interrupted" if it is interrupted for the longest time t_0 . Here $t_0 = \arg \max_{l \in L_k} t_l$, where t_l is defined as $y_{l,t} = 0, t = 1, \dots, t_l, y_{lt} \geq 1, t = t_l + 1, \dots, T$ and L_k is a set of arcs belonging to the loop k .

If $T \leq 2$, then the optimization at each stage can be carried out by a simple comparison of all corresponding states. If T is greater, then some dynamic programming procedure is usually more efficient.

The optimization stops, if the cost of net changes is less than ϵ during K steps of global line-search.

The CPU time τ of global line-search software developed by J.Valevičienė can be estimated as

$$\tau = cKINM.$$

Here c depends on a computer, for PC approximately $c = 0.2 - 1.0 \text{ sec.}$, K is the number of loops, I is the number of nodes, N is the number of flow components and M is the average number of states of the arcs.

Since 1969 the algorithm was used for optimal planning of the North-Western power system of the former USSR in designing new power transmission lines of 110 KV, 220 KV and 330 KV in the Leningrad branch of "Energosetprojekt", which was the leading institution in the country at the time. These results indicates that the global line-search is a good heuristic that may be improved by randomization and parameters optimization in the framework of the BHA.

The exact dynamic programming procedures were also used at the same place, but with a lesser success. The reason was that the approximate global line-search method was solving the problems up to 100 nodes and more. The exact

dynamic programming procedures were directly applicable only to problems with tens of nodes.

Pardalos and Rosen [119] present an approximation technique based on piecewise linear underestimation of concave cost functions $f_j(x_j)$. The resulting model is a linear, zero-one, mixed integer problem. A direct comparison of this approach and global line-search techniques is an interesting problem of future research. For a review of results in network optimization see [57].

9

SOLVING DIFFERENTIAL EQUATIONS BY EVENT-DRIVEN TECHNIQUES FOR PARAMETER OPTIMIZATION

9.1 INTRODUCTION

Here the optimization of parameters of a large system of non-linear differential equations is considered. The system is assumed to simulate semiconductor circuits. We apply the well known event-driven techniques to obtain an approximate solution fast. We extend these techniques by considering pairs of nodes, instead of single nodes. The "twin-node" technique is more efficient in tightly coupled circuits as compared with the "single-node" one. The improvement of efficiency of the solution is important in optimizing the parameters of circuits by the global optimization techniques including BHA.

9.2 EVENT-DRIVEN TECHNIQUES

In the circuit simulation, we represent each of non-linear devices in the circuit by a time-invariant network with non-linear resistors, non-linear capacitors, and non-linear current sources. Using nodal formulation [20] we obtain:

$$C(x)\dot{x} + G(x)x = U(t), \quad (9.1)$$

where x represents voltages of n nodes, $C(x) = (c_{ij}(x)), i, j = 1, \dots, n$ is the matrix of non-linear capacitors, $G(x) = (g_{ij}(x)), i, j = 1, \dots, n$ is the matrix of non-linear conductances, and $U(t) = (u_i(t)), i = 1, \dots, n$ represents a time-

dependent input. The order of equation (9.1) is assumed equal to the number of nodes n .

Non-linear differential equation (9.1) is represented as a sequence of linear differential equations. We do it by fixing the voltage-dependent parameters $C(x^k) = C^k$ and $G(x^k) = G^k$ at some discrete values $x = x^k$, corresponding to the moments $t = t_k$. Time-dependent parameters $U(t_k) = U^k$ are fixed at the moments $t = t_k$. We update those parameters after each step $k = 0, 1, 2, \dots$ of the approximate solution. Thus for any node i we can write

$$c_{ii}^k \frac{dx_i}{dt} + g_{ii}^k x_i - \sum_{j \in J_i} c_{ij}^k \frac{dx_j}{dt} - \sum_{j \in J_i} g_{ij}^k x_j = u_i^k, i = 1, \dots, n, k = 0, 1, 2, \dots, K. \quad (9.2)$$

Here J_i is a set of nodes adjacent to node i , $c_{ij} = c_{ji}$, and $g_{i,j} = g_{j,i}$. Increasing the number of steps K linear solution (9.2) may approach the solution of non-linear equations (9.1).

The popular system of semiconductor circuit simulation SPICE [108] uses sparse matrix techniques [25], while solving the equations similar to (9.2). Thus, the results of simulation may converge to the exact solution of system (9.1). However, the amount of calculations using SPICE is too large to carry out the multiple simulation, which is usually needed, if we wish to optimize the circuit.

The alternative system CINNAMON, approximately solving equations (9.2), uses event-driven technique [47, 155]. The technique considers only one node at a time, keeping the remaining nodes "constant". This means that in system (9.2) the voltages x^k and their derivatives \dot{x}^k at the adjacent nodes remain constant until the next $k + 1$ step.

The CINNAMON system defines the next moment t^{k+1} as the minimal time for obtaining a significant change in voltage. This means that we fix a set of voltage values x^k , but not the time moments t_k , like in traditional techniques. Thus the next moment t^{k+1} is the nearest moment when the voltage reaches the next (higher or lower) discrete value x_j^k . We define this moment as an event. The event $k + 1$ means that we update values of $x = x^{k+1}$, $\dot{x} = \dot{x}^{k+1}$, $C = C^{k+1}$, and $G = G^{k+1}$.

One may obtain the next event t^{k+1} by solving linear differential equation (9.2) for each node $i = 1, \dots, n$. In this way we define the "critical" moment t_i^k when

the nearest discrete value of the voltage x_i^{k+1} is reached. We can do that by analytical solution of (9.2) for $i = 1, \dots, n$. Then

$$t^{k+1} = \min_{1 \leq i \leq n} t_i^{k+1}. \quad (9.3)$$

If there are no tightly coupled nodes, then this technique usually works very well. However we get a lot of trouble, if there is at least a pair of tightly coupled nodes, what is rather usual.

The main idea of this paper is to extend the event-driven techniques to the case of tightly coupled nodes. We can do that by considering separately not single nodes but pairs of nodes. We may obtain the analytical solution of the pairs (i, j) of linear differential equations (9.2). Thus one can define the "critical" moments t_i^k for each node i and may obtain the next event by expression (9.33).

Obviously the Twin-Node (TN) algorithm is more complicated as compared to the Single-Node (SN) one. However it pays, if some nodes are tightly coupled. We shall describe the TN algorithms later, in Section 9.4

The TN event-driven techniques usually work well, but need more calculations. We can reduce the unnecessary calculations, if we consider not all pairs of adjacent nodes but only tightly coupled pairs. Thus the Twin-Node algorithm turns into a Single-Node one of CINNAMON type, if there are no tightly coupled nodes.

9.3 CRITICAL MOMENT

We may define critical moments $t_i^{k+1} = t_i^{k+1}(j)$ by the analytical solution of the pair (i, j) of linear differential equations (9.2), keeping the remaining nodes "constant". This is the main part of the Twin-Node algorithm, therefore we shall give a complete description. From expression (9.2), writing all constants on the right side, we obtain:

$$\begin{aligned} c_{ii}^k \frac{dx_i}{dt} + g_{ii}^k x_i - c_{ij}^k \frac{dx_j}{dt} - g_{ij}^k x_j &= \sum_{l \in J_i \setminus j} c_{il}^k \frac{dx_l}{dt} + \sum_{l \in J_i \setminus j} g_{il}^k x_l + u_i^k \\ c_{jj}^k \frac{dx_j}{dt} + g_{jj}^k x_j - c_{ji}^k \frac{dx_i}{dt} - g_{ji}^k x_i &= \sum_{l \in J_j \setminus i} c_{jl}^k \frac{dx_l}{dt} + \sum_{l \in J_j \setminus i} g_{jl}^k x_l + u_j^k. \end{aligned} \quad (9.4)$$

Denote the right sides by b_i and b_j , respectively

$$\begin{aligned} c_{ii}^k \frac{dx_i}{dt} + g_{ii}^k x_i - c_{ij}^k \frac{dx_j}{dt} - g_{ij}^k x_j &= b_i \\ c_{jj}^k \frac{dx_j}{dt} + g_{jj}^k x_j - c_{ji}^k \frac{dx_i}{dt} - g_{ji}^k x_i &= b_j, \end{aligned} \quad (9.5)$$

where

$$\begin{aligned} b_i &= \sum_{l \in J_i \setminus j} c_{il}^k \frac{dx_l}{dt} + \sum_{l \in J_i \setminus j} g_{il}^k x_l + u_i^k \\ b_j &= \sum_{l \in J_j \setminus i} c_{jl}^k \frac{dx_l}{dt} + \sum_{l \in J_j \setminus i} g_{lj}^k x_l + u_j^k \end{aligned} \quad (9.6)$$

The coefficients A_i, A_j and \bar{A}_i, \bar{A}_j of partial solutions are defined by the following equations

$$\begin{aligned} (c_{ii}r + g_{ii})A_i - (c_{ij}r + g_{ij})A_j &= 0 \\ -(c_{ji}r + g_{ji})A_i + (c_{jj}r + g_{jj})A_j &= 0 \\ (c_{ii}\bar{r} + g_{ii})\bar{A}_i - (c_{ij}\bar{r} + g_{ij})\bar{A}_j &= 0 \\ -(c_{ji}\bar{r} + g_{ji})\bar{A}_i + (c_{jj}\bar{r} + g_{jj})\bar{A}_j &= 0. \end{aligned} \quad (9.7)$$

The eigen-values r and \bar{r} in (9.7) are defined by the characteristic equation

$$\begin{vmatrix} c_{ii}r + g_{ii} & -c_{ij}r - g_{ij} \\ -c_{ji}r - g_{ji} & c_{jj}r + g_{jj} \end{vmatrix} = 0. \quad (9.8)$$

Solving equation (9.8) we obtain:

$$r = -p/2 + \sqrt{p^2/4 - q + \epsilon} \quad (9.9)$$

$$\bar{r} = -p/2 - \sqrt{p^2/4 - q + \epsilon}, \quad (9.10)$$

where

$$p = \frac{g_{ii}c_{jj} + g_{jj}c_{ii} - c_{ij}g_{ij} - c_{ji}g_{ji}}{c_{ii}c_{jj} - c_{ij}c_{ji}} \quad (9.11)$$

$$q = \frac{g_{ii}g_{jj} - g_{ij}^2}{c_{ii}c_{jj} - c_{ij}^2}. \quad (9.12)$$

Here $\epsilon > 0$ is a small number that keeps r and \bar{r} different to avoid unnecessary complications.

Substituting r and \bar{r} from expressions (9.9) and (9.10) into expression (9.7) we can write

$$A_j = \frac{c_{ii}r + g_{ii}}{c_{ij}r + g_{ij}} A_i = \frac{c_{ji}r + g_{ji}}{c_{jj}r + g_{jj}} A_i \quad (9.13)$$

$$\bar{A}_j = \frac{c_{ii}\bar{r} + g_{ii}}{c_{ij}\bar{r} + g_{ij}} \bar{A}_i = \frac{c_{ji}\bar{r} + g_{ji}}{c_{jj}\bar{r} + g_{jj}} \bar{A}_i. \quad (9.14)$$

Now the partial solutions of homogeneous equations (9.5)

$$\begin{aligned} x_i &= A_i e^{rt} = D_i e^{rt} \\ x_j &= A_j e^{rt} = h D_i e^{rt}, \end{aligned} \quad (9.15)$$

and

$$\begin{aligned} x_i &= \bar{A}_i e^{\bar{r}t} = D_j e^{\bar{r}t} \\ x_j &= \bar{A}_j e^{\bar{r}t} = \bar{h} D_j e^{\bar{r}t}. \end{aligned}$$

Here D_i and D_j are integration constants and

$$\begin{aligned} h &= \frac{c_{ii}r + g_{ii}}{c_{ij}r + g_{ij}} = \frac{c_{ji}r + g_{ji}}{c_{jj}r + g_{jj}} \\ \bar{h} &= \frac{c_{ii}\bar{r} + g_{ii}}{c_{ij}\bar{r} + g_{ij}} = \frac{c_{ji}\bar{r} + g_{ji}}{c_{jj}\bar{r} + g_{jj}}. \end{aligned}$$

Thus, the general solution of homogeneous equations (9.5) is

$$\begin{aligned} x_i &= D_i e^{rt} + D_j e^{\bar{r}t} \\ x_j &= h D_i e^{rt} + \bar{h} D_j e^{\bar{r}t}. \end{aligned} \quad (9.16)$$

We can obtain a partial solution of non-homogeneous equations (9.5) by substituting the general solution x_i and x_j (9.16) of the corresponding homogeneous part of (9.5) into complete non-homogeneous equations (9.5), and assuming that D_i and D_j are not constants but depend on time $D_i = D_i(t)$, $D_j = D_j(t)$. Thus

$$\begin{aligned} c_{ii}\dot{D}_i e^{rt} + c_{ii}\dot{D}_j e^{\bar{r}t} - c_{ij}h\dot{D}_i e^{rt} - c_{ij}\bar{h}\dot{D}_j e^{\bar{r}t} &= b_i \\ -c_{ji}\dot{D}_i e^{rt} - c_{ji}\dot{D}_j e^{\bar{r}t} + c_{jj}h\dot{D}_i e^{rt} + c_{jj}\bar{h}\dot{D}_j e^{\bar{r}t} &= b_j \end{aligned} \quad (9.17)$$

or

$$\begin{aligned} (c_{ii} - c_{ij}h)\dot{D}_i e^{rt} + (c_{ii} - c_{ij}\bar{h})\dot{D}_j e^{\bar{r}t} &= b_i \\ (-c_{ji} + c_{jj}h)\dot{D}_i e^{rt} + (-c_{ji} + c_{jj}\bar{h})\dot{D}_j e^{\bar{r}t} &= b_j. \end{aligned} \quad (9.18)$$

Here $\dot{D}_i = dD_i(t)/dt$, and $\dot{D}_j = dD_j(t)/dt$. Denote

$$a_i = c_{ii} - c_{ij}h, \quad \bar{a}_i = c_{ii} - c_{ij}\bar{h} \quad (9.19)$$

$$a_j = -c_{ji} + c_{jj}h, \quad \bar{a}_j = -c_{ji} + c_{jj}\bar{h}, \quad (9.20)$$

and

$$\begin{aligned} v_i &= a_i e^{rt}, \quad \bar{v}_i = \bar{a}_i e^{\bar{r}t} \\ v_j &= a_j e^{rt}, \quad \bar{v}_j = \bar{a}_j e^{\bar{r}t}. \end{aligned} \quad (9.21)$$

Then from expression (9.18)

$$\begin{aligned} \dot{D}_i v_i + \dot{D}_j \bar{v}_i &= b_i \\ \dot{D}_i v_j + \dot{D}_j \bar{v}_j &= b_j. \end{aligned} \quad (9.22)$$

Expression (9.22) yields

$$\begin{aligned} \dot{D}_i &= \frac{b_i \bar{v}_j - b_j \bar{v}_i}{v_i \bar{v}_j - v_j \bar{v}_i} \\ \dot{D}_j &= \frac{b_i v_j - b_j v_i}{\bar{v}_i v_j - \bar{v}_j v_i} \end{aligned} \quad (9.23)$$

From expressions (9.23) and (9.21)

$$\begin{aligned} \dot{D}_i &= \frac{b_i \bar{a}_j - b_j \bar{a}_i}{a_i \bar{a}_j - a_j \bar{a}_i} e^{-rt} \\ \dot{D}_j &= \frac{b_i a_j - b_j a_i}{\bar{a}_i a_j - \bar{a}_j a_i} e^{-\bar{r}t}. \end{aligned} \quad (9.24)$$

Denote

$$\begin{aligned} H_i &= \frac{b_i \bar{a}_j - b_j \bar{a}_i}{a_i \bar{a}_j - a_j \bar{a}_i} \\ H_j &= \frac{b_i a_j - b_j a_i}{\bar{a}_i a_j - \bar{a}_j a_i}. \end{aligned} \quad (9.25)$$

Then

$$\begin{aligned} \dot{D}_i &= H_i e^{-rt} \\ \dot{D}_j &= H_j e^{-\bar{r}t}. \end{aligned} \quad (9.26)$$

After integration of expression (9.26)

$$\begin{aligned} D_i &= \begin{cases} -1/r H_i e^{-rt} + D_{i0}, & \text{if } r \neq 0 \\ H_i t + D_{i0}, & \text{if } r = 0 \end{cases} \\ D_j &= \begin{cases} -1/\bar{r} H_j e^{-\bar{r}t} + D_{j0}, & \text{if } \bar{r} \neq 0 \\ H_j t + D_{j0}, & \text{if } \bar{r} = 0 \end{cases}. \end{aligned} \quad (9.27)$$

From expressions (9.16) and (9.27)

$$\begin{aligned}x_i &= -(H_i/r + H_j/\bar{r}) + D_{i0}e^{rt} + D_{j0}e^{\bar{r}t} \\x_j &= -(hH_i/r + \bar{h}H_j/\bar{r}) + D_{i0}e^{rt} + D_{j0}e^{\bar{r}t}.\end{aligned}\quad (9.28)$$

We are looking for a partial solution, therefore we set zero integration constants $D_{i0} = D_{j0} = 0$. Then the partial solution of non-homogeneous equations (9.5) is

$$\begin{aligned}x_i &= -(H_i/r + H_j/\bar{r}) \\x_j &= -(hH_i/r + \bar{h}H_j/\bar{r}).\end{aligned}\quad (9.29)$$

Expressing the general solution of non-homogeneous equation (9.5) as a sum of general solution (9.16) of homogeneous equation (9.5) and the partial solution (9.29) of non-homogeneous equation (9.5), we obtain

$$\begin{aligned}x_i &= -(H_i/r + H_j/\bar{r}) + D_i e^{rt} + D_j e^{\bar{r}t} \\x_j &= -(hH_i/r + \bar{h}H_j/\bar{r}) + hD_i e^{rt} + \bar{h}D_j e^{\bar{r}t}.\end{aligned}\quad (9.30)$$

Here D_i and D_j are constants. Let us define these constants by the initial values of x . Denote the values of x_i and x_j at the zero moment $t = 0$ as x_{i0} and x_{j0} , respectively.

Denote

$$\begin{aligned}H &= -(H_i/r + H_j/\bar{r}) \\ \bar{H} &= -(hH_i/r + \bar{h}H_j/\bar{r})\end{aligned}$$

Then, the initial values

$$\begin{aligned}x_{i0} &= H + D_i + D_j \\x_{j0} &= \bar{H} + hD_i + \bar{h}D_j.\end{aligned}\quad (9.31)$$

We can define constants D_i and D_j by solving equations (9.31)

$$\begin{aligned}D_i &= \frac{(H - x_{i0})\bar{h} - (\bar{H} - x_{j0})}{h - \bar{h}} \\D_j &= \frac{(H - x_{i0})\bar{h} + (\bar{H} - x_{j0})}{h - \bar{h}}.\end{aligned}\quad (9.32)$$

9.4 TWIN-NODE EVENT-DRIVEN TECHNIQUES

Let us specify the order in which nodes will be processed. We make as few changes as possible in comparison to the well known Single-Node Event-Driven technique CINNAMON. First recall the CINNAMON algorithm:

Step 0 : Set the starting time $t = t^0$. Linearize equations at the starting voltage level $x_i = x_i^0$.

Step 1 : Solve equations (9.2) for each node i . Determine the time t_i^1 when the node i marks the next event ¹.

Step 2 : Choose a node i^1 with the smallest time $t^1 = \min_i t_i^1$ and set the respective voltage to the new level $x_{i^1} = x_{i^1}^1$.

Step 3 : Linearize the equations at the new voltage level $x_{i^1}^1$ for all the nodes i adjacent to the node i^1 .

Step 4 : Set the time for this group of nodes to t^1 and return to *Step 2*.

The main idea of CINNAMON is that the perturbation induced in some node causes perturbations in the adjacent nodes and thus propagates through the circuit. We take advantage of this wave-like behavior of the algorithm in the Twin-Node technique, too. We refer to the TN technique as CINNAMON2.

Suppose that an event occurred in the node i . In the SN technique, all the voltage values and their derivatives of the adjacent nodes are considered constant. We solve a single equation for each single node i and thus define the next event time t_i^{k+1} .

Using TN technique we solve the systems of two equations for the pairs of the node i and all the adjacent nodes j . Denote by t_i^{k+1} the time when the voltage of the event node i reaches the next discrete level.

The event time t_i^{k+1} for the fixed event node i depends on the "partner" node j , too. This means that $t_i^{k+1} = t_i^{k+1}(j)$. We choose the smallest $t_i^{k+1}(j)$ as the next event time t^{k+1} .

¹The moment when voltage x_i will reach the next discrete level.

$$t^{k+1} = \min_j t_i^{k+1}(j). \quad (9.33)$$

Now let us describe the Twin-Node algorithm:

Step 0 : Set the starting time $t = t^0$. Linearize the equations at the starting voltage level $x_i = x_i^0$.

Step 1 : Solve equations (9.2) for each node i . Determine the time t_i^1 when the node i marks the next event

Step 2 : Choose a node i^1 with the smallest time $t^1 = \min_i t_i^1$ and set the respective voltage to the new level $x_{i^1} = x_{i^1}^1$.

Step 3 : Linearize the equations at the new voltage level $x_{i^1}^1$ for all the nodes i adjacent to the node i^1 .

Step 4 : Define the times $t_i^1(j)$ corresponding to each adjacent node j by solving the pairs (i, j) of equations (9.2).

Step 5 : Choose for time t^1 the smallest of $t_i^1(j)$

Step 6 : Set the time t^1 for the node i and all the adjacent nodes j . Return to *Step2*.

9.5 COMPUTING RESULTS

We consider three techniques:

- Single-Node CINNAMON [155];
- Twin-Node CINNAMON2;
- Multi-Node SPICE that solves the complete system (9.1) directly using the sparse matrix techniques [25, 159].

These techniques are compared using three examples.

Example 1: A circuit (Fig 9.1) of two RC elements, where $C1 = C2 = 1$, and $R1 = 500$, $R2 = 1$, $V = 5$. Fig 9.2 shows how the voltage of node 3 depends on time. Using the CINNAMON algorithm we see some voltage delay. The results of both CINNAMON2 and SPICE algorithms are close enough.

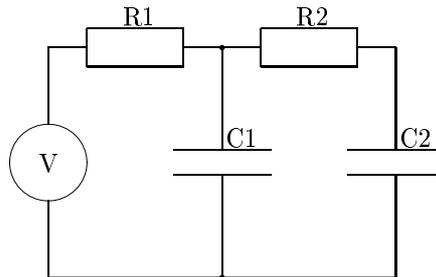


Figure 9.1 The circuit of two RC elements

Example 2: A circuit of ten RC elements, where $C1 = C2 = \dots = C10 = 1$, $R1 = R2 = \dots = R8 = 10$, $R9 = 100$, $R10 = 1$, and $V = 5$.

Figure 9.3 shows how the voltage of node 11 depends on time. Like in the previous example, CINNAMON voltage demonstrates a delay. The remaining two algorithms show very close results.

Example 3: The same circuit as in example 2, but resistances are different: $R1 = R3 = R5 = R7 = R9 = 1000$, $R2 = R4 = R6 = R8 = R10 = 1$, $V = 5$. Figure 9.4 shows the voltage of node 11. We see that the Twin-Node CINNAMON2 and the Single-Node CINNAMON don't reach the stable voltage level. The example shows that with a strong coupling between serial nodes the results of the CINNAMON2 algorithm are between SPICE and CINNAMON algorithms, but closer to SPICE.

The results show that TN solutions of systems of non-linear differential equations are good heuristics and thus may be used applying BHA for the optimization of circuits parameters. It is useful to complete the circuit optimization by the "exact" Multi-Node techniques at the optimized circuit parameters.

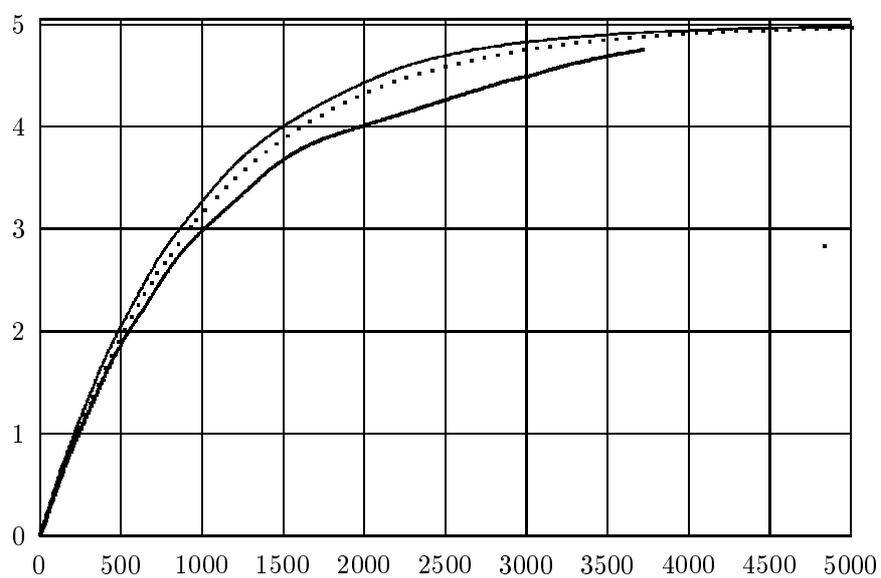


Figure 9.2 Comparison of voltages by three techniques in the circuit of two RC elements and a weak coupling. A thin line is obtained by CINNAMON, a thick line is by CINNAMON2, and a dotted line is by SPICE

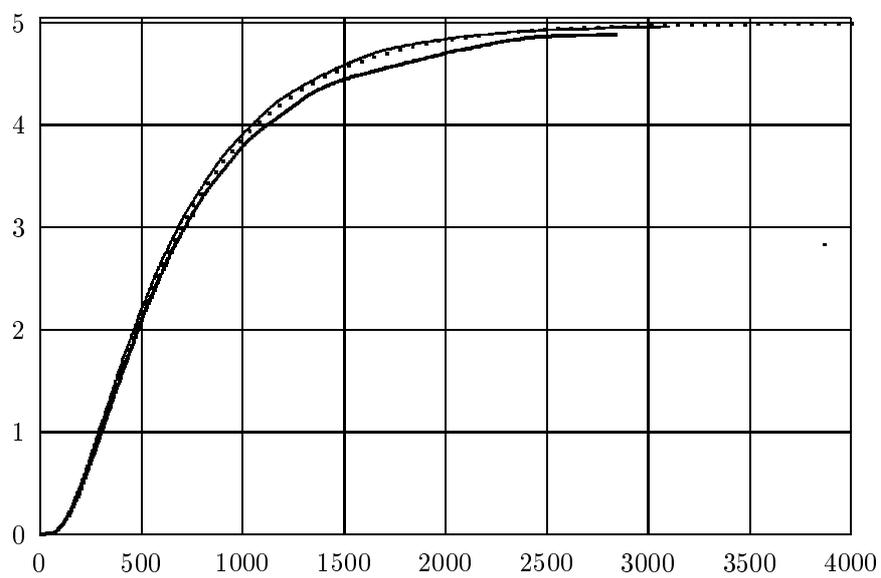


Figure 9.3 Comparing voltages by three techniques in the circuit of ten RC elements and a weak coupling

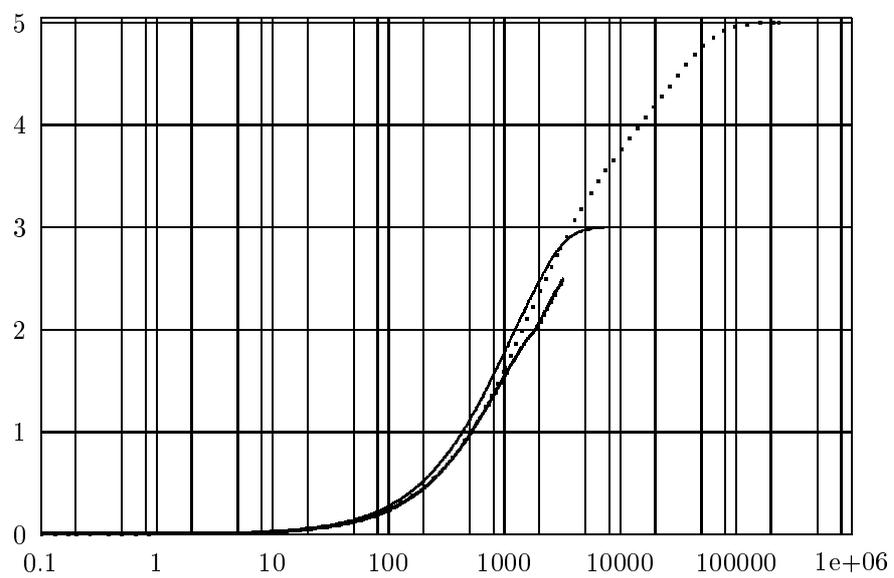


Figure 9.4 Comparing voltages by three techniques in the circuit of ten RC elements and a strong coupling

10

OPTIMIZATION IN NEURAL NETWORKS

10.1 INTRODUCTION

The idea of Artificial Neural Networks (ANN) is to build a massively connected and highly parallel system from simple processing units. This idea stems from real biological systems. These units are regarded as simplification of biological neurons what explains the ANN term. The information in ANN is stored in "weights" of connections between the units. It is assumed that ANN "adapts" to a problem by changing these weights.

ANN are nonlinear systems of large amount of parameters (up to $10^3 - 10^4$). An additional difficulty is that the optimization of neural networks is a multimodal problem, so the global optimization methods are needed. An astonishing moment is that the local optimization techniques are widely used in ANN and yield satisfactory results in many cases.

Thus it is important to know why the local optimization is so successful in optimizing neural networks. In this chapter, we try to get an answer considering a perceptron-like unit which is the building block for most ANN architectures. The reason seems to be that by optimizing weights of a linear approximation of a neural network one can provide a good starting point for the local optimization (see [139, 140]). It means that the solution of a linear problem may be regarded as a good heuristic for the non-linear one. This may be helpful in developing faster and more robust algorithms for the optimization of Artificial Neural Network weights using the Bayesian Heuristic Approach.

The new ANN "training" technique (so-called *backward error propagation* or *back-propagation* method) [133] made ANN a useful tool for solving real world

problems. This training method may be regarded as a stochastic gradient algorithm (often called as stochastic approximation [132]). Therefore the convergence rate of the back-propagation algorithm is as slow as that of a stochastic gradient (see [146, 39]).

Numerical optimization techniques offer a set of so-called second-order methods providing the super-linear convergence rate (see expression (3.8)). The conjugate gradient method is used in ANN optimization [71]. The variable metrics method [153] is less popular in ANN, apparently because using the variable metrics one needs to keep and update a high order matrix. However, the conjugate gradient accumulates calculation errors.

ANN may "learn" to perform a specific task by presenting examples of desired mapping to the network. The "weights" are adjusted to minimize the total sum of errors between the actual and the desired mapping. Since the optimization of ANN is multi-modal, the local optimization techniques can fail. Theoretically the global optimization methods should be preferable in this situation, However, in practice, the local optimization technique often yields reliable results. This contradiction indicates some specific features of ANN. The success of local optimization in ANN can be explained by a good initialization of weights, because any local search procedure depends on initial conditions .

Therefore we ought to know error surfaces of ANN while designing efficient learning algorithms. The error surface illustrates how the objective function depends on on the ANN weights.

The objective of this chapter is to show the error surfaces of different simple ANN (including the feed-forward neural networks). We discuss why and how local search procedures perform on those surfaces (see [139, 140]). We do not give any recommendations or rules how to improve the learning in ANN, but just present and discuss some relevant illustrations. Our objective is to understand why the local search works in multi-modal ANN optimization problems. This way we define a family of promising heuristics applying BHA to ANN optimization.

10.2 ERROR SURFACES

The following three important features of error surfaces are discussed in [67].

- if the number of training samples is small, then the error surface is like a step-function, where steps correspond to individual samples;
- the surface becomes smoother increasing the number of training samples;
- there are long narrow ravines such that are flat in one direction and steep in other directions.

These characteristics are not convenient for the standard local search techniques. It is not clear how to apply the gradient algorithms in optimizing step-functions because the gradients of the "flat" regions are zero. That and the multi-modality make it difficult to understand why the local search procedures work. Further we discuss the factors explaining the success of local search procedures considering some examples of the error surfaces of the feed-forward neural networks [140].

10.3 PERCEPTRON AND LINEAR DISCRIMINANT

It is difficult to characterize precisely the error surface without a proper visualization. However, the visualization is complicated in dimensions higher than three. Thus we start by exploring a simple example of a single-node network. This example is important, because many of single-node characteristics appear in more complicated ANN, including hidden nodes. An example of the feed-forward ANN with a hidden layer is shown in Figure 10.1.

A primary building block for most ANN architectures is a Perceptron unit, the output of which is

$$y = \varphi\left(\sum_{i=0}^K w_i x_i\right), \quad (10.1)$$

where $\{w_0, \dots, w_K\}$ are adjustable weights (parameters) of the unit, and $\{x_0, \dots, x_K\}$ is a $K + 1$ dimensional input vector with $x_0 = 1$.

The weight w_0 on this extra input is called a bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights. The activation

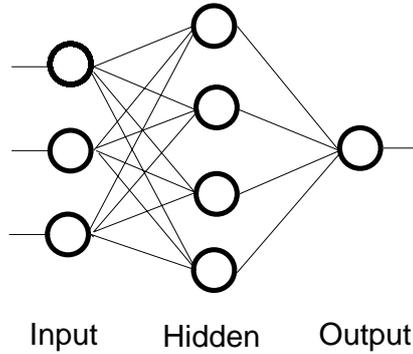


Figure 10.1 Architecture of the feed-forward ANN with one hidden layer. Solid circles denote processing units, straight lines are connections.

function $\varphi(\cdot)$ is nonlinear and monotone. There is a strong relationship between the Perceptron unit (10.1) and pattern recognition.

A concept central to the pattern recognition is that of discriminant. The idea is that a pattern recognition system learns adaptively from the experience and constructs an optimal discriminant function $D(x)$ defining decision boundaries.

A popular Perceptron discriminant is a linear function

$$D(x) = \sum_{i=0}^K w_i x_i. \quad (10.2)$$

In more general cases the discriminant could be a non-linear function of the weight vector $\{w_0, \dots, w_K\}$ and the input vector $\{x_0, \dots, x_K\}$

$$D(x) = D(w, x). \quad (10.3)$$

In the case of a two classes A and B , the task is to get weight values such that all the patterns could be classified as correctly as possible:

$$\begin{aligned} D(x) &> 0, & \text{iff } x \in A \\ D(x) &< 0, & \text{iff } x \in B. \end{aligned} \quad (10.4)$$

Here the line $D(x) = 0$ is a decision boundary.

In general, to solve problem (10.4) one should:

- assign *target* values for the classes, for example, +1 for the class A and -1 for the class B ;
- define an error depending on the distance between the classifier output $y = D(x)$ and the target value, for example, the squared Euclidean distance;
- compose an *objective function* as a sum of errors over the training data points $x(n)$, $n = 1, \dots, N$;
- search for weights w of the discriminant function $D(w, x)$ such that minimizes the objective:

$$\min_w \frac{1}{N} \sum_{n=1}^N (y(x(n)) - t(n))^2. \quad (10.5)$$

Here $t(n)$ is the target (desired output value) at the data point $x(n)$, for example, $t(n) = -1$ for the class A , and $t(n) = 1$ for the class B , and $y(x(n))$ is the output of a classifier for the data point $x(n)$.

Using a linear discriminant function (see expression (10.2)), task (10.4) reduces to quadratic programming problem (10.5) which transforms to a set of linear equations, corresponding to partial derivatives of sum (10.5). The Perceptron is the classical example of linear discriminant functions.

The activation function φ shows how the error of a classifier is defined. The linear function ($\varphi(x) \equiv x$) means that the error is defined as a squared distance between the actual output and the target value (see expression (10.5)). In this case, the classifier is rather sensitive to outliers, e.g., only one specific data point drawn from the tail of the distribution can significantly shift the resulting linear discriminant.

Using a "hard-limiting" step-function φ , we just count the number of misclassifications exceeding the limit error. The error-counting classifier is not sensitive to outliers but it ignores all the misclassifications which do not reach the "hard-limit".

A reasonable compromise between the linear and the hard-limit activation functions is a smooth monotone activation function φ defining an outliers-insensitive classifier, capable of weighting the classification error, depending on the distance between the output and target values

We may achieve a similar effect by using linear φ and defining the error in expression (10.5) as an absolute value instead of a squared one (see [94]). The advantage is a possibility to apply the standard methods of linear programming.

The idea is illustrated by the simplest one-layer linear problem. We minimize the sum of absolute deviations

$$\begin{aligned} \min_w 1/N \sum_{n=1, N} |s^n| \\ s_n = t(n) - y^n \\ y^n = 1/m \sum_{j=1, m} x_j^n w_j. \end{aligned}$$

Here y^n is the output and the vector $x^n = (x_j^n, j = 1, \dots, m)$ is the input of the learning set $n = 1, \dots, N$. We optimize the weight vector $w = (w_j, j = 1, \dots, m)$ to get the minimum of a sum of absolute deviations. It is assumed that $x_j, a_j, y^n \in [0, 1]$. By introducing auxiliary non-negative variables u_n we reduce piece-wise linear problem (10.5) to the one of linear programming:

$$\begin{aligned} \min_{w, u} \sum_{n=1, N} (u^n) \\ u_n \geq s_n \\ u_n \geq -s_n \\ s_n = t^n - y^n \\ y^n = 1/m \sum_{j=1, m} x_j^n w_j. \end{aligned} \tag{10.6}$$

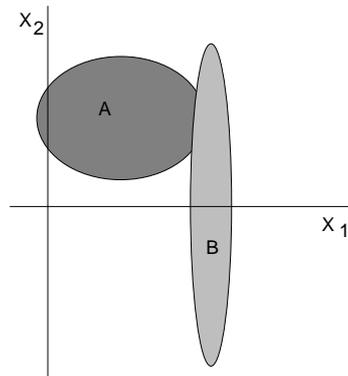


Figure 10.2 Two overlapping Gaussian distributions of Highleyman's classes.
 Class A: $\mu = (1, 1)$, $\sigma = (1, 0.5)$, class B: $\mu = (2, 0)$, $\sigma = (0.1, 2)$.

A great variety of ANN architectures can be composed by connecting the outputs of one group of Perceptron-like units with the inputs of another group of units. If the function φ is a "hard-limiting" step-function, then the decision boundary of a composed neural network could be analyzed as a superposition of linear discriminants. Given a smooth activation function φ , ANN can not be considered as a simple superposition of linear discriminants.

An illustration is a feed-forward network composed of a single linear output and two hidden nodes with the activation function $\varphi(\cdot) = \tanh(\cdot)$. The network was trained to separate 10 points on 2D input space into two classes. Figure 10.9 shows that such a network creates complicated decision boundaries. We see that these decision boundaries can not be expressed as a superposition of two linear discriminant.

10.4 TESTING HIGHLEYMAN'S PROBLEM

We will analyze the performance of Perceptron unit (10.1) on the classical example of Highleyman's classes [60]. The set of data consists of two overlapping Gaussian distributions with the mean μ and dispersion σ parameters shown in Figure 10.2.

The overlap of these two classes is 6%, but the optimal result for the linear discriminant function is 10% of errors. For this problem a linear discriminant function is as follows:

$$D(x) = w_o + w_1 x_1 + w_2 x_2. \quad (10.7)$$

The task is from a given set of N points ($N/2$ points belong to the class A, $N/2$ - to the class B) to find the optimal discriminant function from a given set of N points ($N/2$ points belong to the class A, $N/2$ - to the class B) by minimizing the error function (10.5).

Here the hyperbolic tangent is used as an activation function ($\varphi(\cdot) = \tanh(\cdot)$). First we parameterize the linear discriminant function (10.7) in terms of spherical angles α_1, α_2 and radius distance R

$$\begin{aligned} w_o &= R \cos \alpha_1 \\ w_1 &= R \sin \alpha_1 \cos \alpha_2 \\ w_2 &= R \sin \alpha_1 \sin \alpha_2 \end{aligned} \quad (10.8)$$

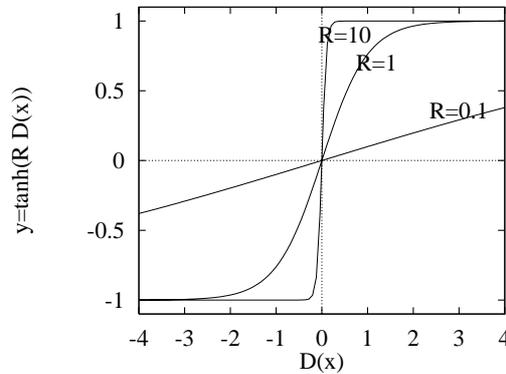


Figure 10.3 Output dependence on the radius parameter R

Obviously, the spherical angles α_1, α_2 define a discriminant line. The radius R defines error weighting (see Figure 10.3) in the following way:

- $R \ll 1$ nearly linear output, squared distance criterion
- $R \sim 1$ nonlinear smooth output function, weighed errors
- $R \gg 1$ nearly "hard-limiting" function, error counting

Error (10.5) as a function of the parameters α_1, α_2, R is a sum of non-convex functions. Thus we may get a multi-modal error function. The objective of this parameterization is to determine how error function (10.5) depends on the radius parameter R and the sample size N .

10.5 ERROR SURFACES OF THE PERCEPTRON UNIT

The influence of factor R is investigated by plotting the 2D projection of (10.5) versus α_1 and α_2 at different fixed R . A comparison for different sample sizes N is also made. The results are presented in Figures 10.4-10.7.

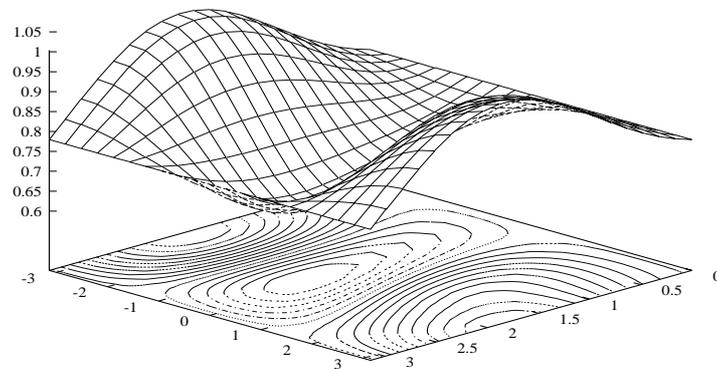


Figure 10.4 The error function at the radius $R = 0.1$, when the sample size $N = 6$.

In the nearly linear case ($R = 0.1$), the error function is a smooth, *unimodal* surface (see Figure 10.4) and the variation of sample size from 6 to 200 makes just a slight change in the surface shape. Increasing the radius parameter R ,

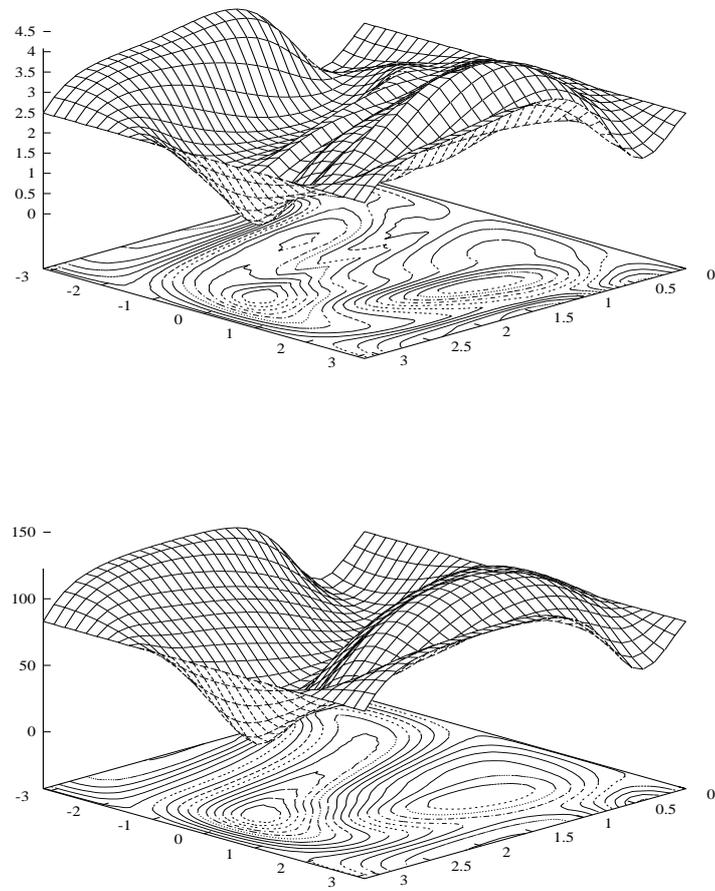


Figure 10.5 The error function at the radius $R = 1$ for the sample size $N = 6$ (top), and $N = 200$ (bottom).

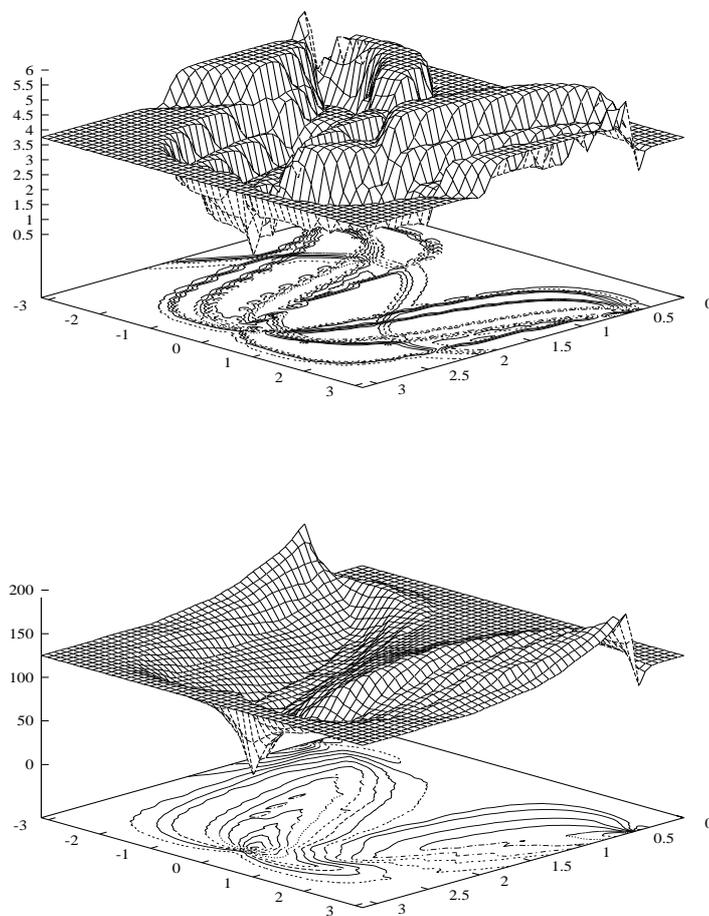


Figure 10.6 The error function at the radius $R = 10$ for the sample size $N = 6$ (top), and $N = 200$ (bottom).

new local minima appear (see the upper pictures in Figures 10.5-10.6). We see a correspondence between individual training samples and the steps on the surface when the the radius parameter R becomes larger. However increasing the sample size N further, the error surface gets smoother, see the bottom pictures in Figures 10.5-10.6. The position of global minimum seems rather insensitive to the changes in R and N . The decision boundary approaches the optimal position early in the learning process and do not changes very much afterwards when one introduces a "non-linearity" by increasing the factor R .

We may control the process by changing the factor R during the optimization. For example, a traditional regularization approach does just that by penalizing the non-smooth regions of the objective function where the weights are large. Figure (10.8) shows the change of weights during the optimization. We see that at the beginning we may optimize the decision boundary by a local search techniques because the system is approximately linear ($R \ll 1$) thus the corresponding error function is smooth and nearly uni-modal. Later the non-linearity appears but the optimal solution often may be found searching for the local minimum along the ravine. On Figure 10.7 we see how the global minimum depends on R . A full picture of the weights dynamics during the training is shown in Figure 10.8.. The weights dynamics follows two phases

- *quasi-linear stage* (up to 8 iterations) when the radius parameter R is small but the angular parameters are close to their the stationary values
- *nonlinear stage* (after 8 iterations) when just the radius parameter is increasing

The last, the nonlinear stage, visually appears like a search for the global minimum along the narrow ravine.

10.6 COMPOSING A NETWORK

The general picture of weight dynamics of a single Perceptron unit not necessarily remains in a complex architecture of ANN consisting (at least) of several hidden units. An extension of single-unit results is not easy because of nonlinear characteristics of the Perceptron unit. For example, it is common to analyze the decision boundaries of a feed-forward network as a superposition of linear discriminants. However, it is the right approach, just in some special cases. For example, if the node activation function ϕ is a "hard limiting" step function.

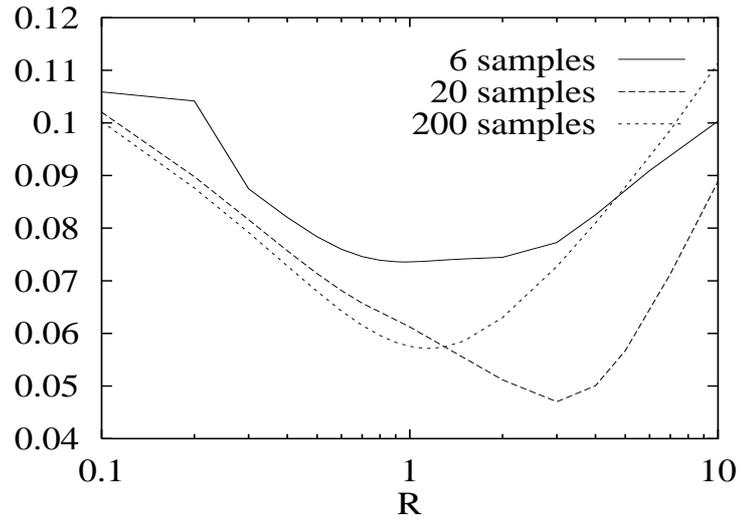


Figure 10.7 Learning error (the depth of global minimum) dependence on the radius parameter R for $N = 200$.

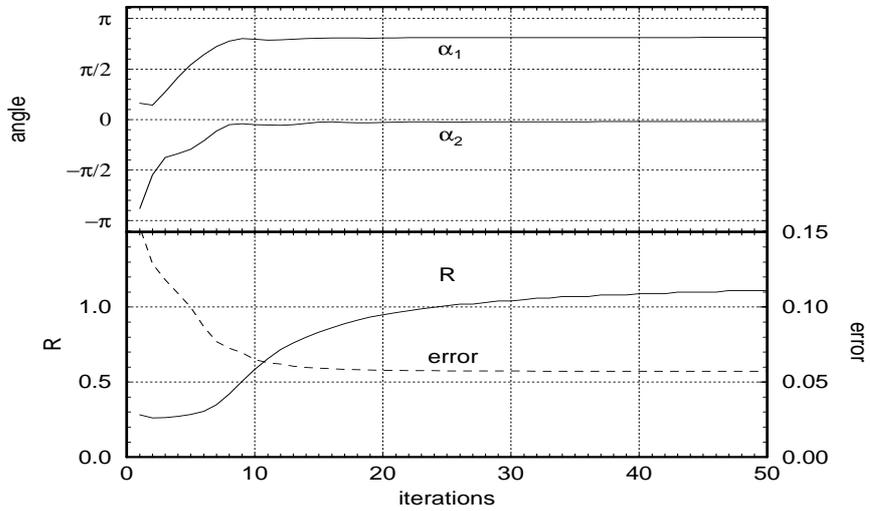


Figure 10.8 A complete dynamics of weights during the training of the Perceptron-like unit.

Given a smooth nonlinear activation function ϕ , feed-forward network cannot

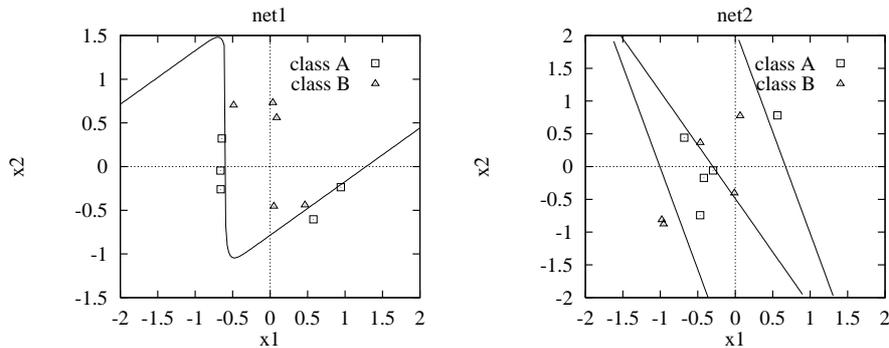


Figure 10.9 Complex decision boundaries of a feed-forward network with two hidden nodes for two different cases *net1* and *net2*.

be considered any longer as a simple superposition of linear discriminants. As an illustration, a feed-forward network, composed of a single linear output and two hidden nodes with the activation function $\varphi(\cdot) = \tanh(\cdot)$, was trained to divide 10 randomly generated points on 2D input space into two classes. It was established that such a network can create more complicated decision boundaries than one may expect (fig.10.9). Moreover, these decision boundaries, as displayed in Figure 10.9, cannot be explained at all as a simple superposition of two linear discriminants.. The examples of functions *net1* and *net2* are as follows:

$$\begin{aligned} \text{net1}(x_1, x_2) = & 5.0453 + 7.0960 \tanh(-0.3348x_1 + 0.5452x_2 - 1.5547) + \\ & 1.7869 \tanh(9.3532x_1 - 1.0599x_2 + 6.3169) \end{aligned} \quad (10.9)$$

$$\begin{aligned} \text{net2}(x_1, x_2) = & 1.2101 - 2.3280 \tanh(1.2583x_1 + 0.4057x_2 + 1.3312) + \\ & 1.0587 \tanh(54.4982x_1 + 32.9479x_2 + 16.7966) \end{aligned} \quad (10.10)$$

The decision boundaries of these networks correspond to zero-level isolines of the functions *net1* and *net2*. The situation illustrates that a feed-forward neural network system cannot be decomposed into functional subunits (neurons) because the aggregate represents more than a mere sum of the components.

10.7 LEARNING WEIGHTS DYNAMICS

A straightforward analysis of error surfaces of a feed-forward network is not easy because of the great number of network parameters. One may visually

observe two-dimensional cross-sections of multi-dimensional parameter space by varying only two parameters and keeping all the others fixed. A disadvantage is that some important features can be undetected examining just two-dimensional cross-sections.

Therefore we use an indirect technique to determine the properties of the error surface by following the dynamics of weights. For each hidden unit i , evolution of the radius parameter $R^{(i)}$ and angles $\alpha_1^{(i)}, \alpha_2^{(i)}$ is observed according to (10.8). The output weights between the hidden units and the network output are not taken into account since, these weights span a linear subspace (the network output is a linear composition of the hidden units). If the network is learning smoothly, avoiding long discontinuous jumps in the parameter space, then the dynamics of network weights can be very informative while analyzing the error surfaces.

We apply more robust gradient-based local search techniques because the conditions for the application of second-order methods are not satisfied here. We use a standard back-propagation algorithm [133]. The learning step was set 0.05, and the momentum term was 0.8.

A two-dimensional test function [68]:

$$f(x, y) = 1.9(1.35 + e^x \sin(13(x - 0.6)^2)e^{-y} \sin(7y)) \quad (10.11)$$

was used to evaluate the weight dynamics of a feed-forward network consisting of a linear composition of 5 Perceptron-like units (10.1) with a hyperbolic tangent activation function. The feed-forward network was learning to approximate the test function (10.11).

The function approximation problem, in general, is more difficult than the classification problem (see Section 10.4). A 3-dimensional perspective of function (10.11) together with the approximation by the feed-forward network, consisting of 5 hidden units, is shown in Figure 10.10. The learning set for a feed-forward network was generated from 225 sample points on a regular grid over $[0, 1]^2$.

The approximation shown in Figure 10.10, is not very accurate. To approximate this test function well a network with ten hidden units would be needed. We use only five hidden units. However, the accuracy seems sufficient for the investigation of the weight dynamics.

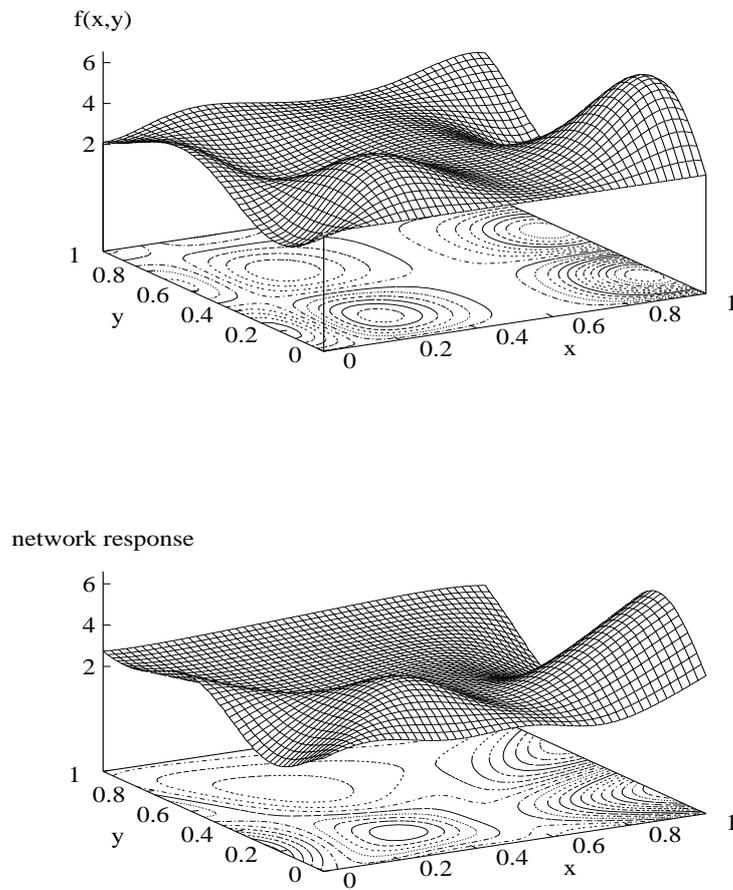


Figure 10.10 Test function (10.11) (top), and the approximation by the feed-forward network with 5 hidden units (bottom).

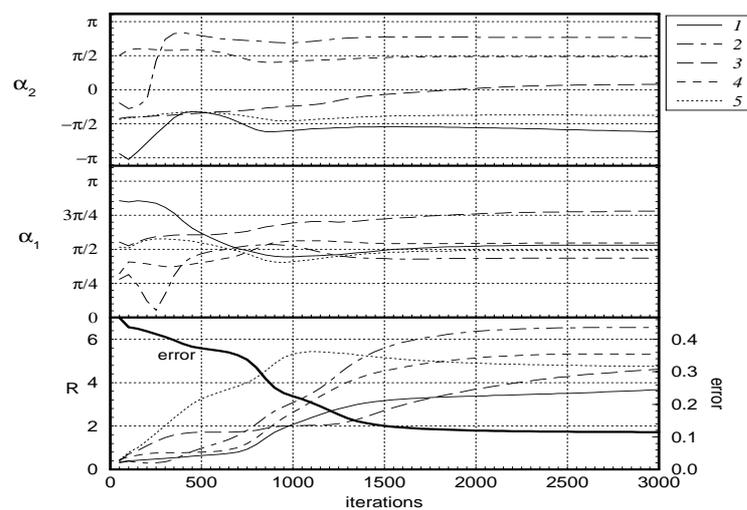


Figure 10.11 Dynamics of the hidden node weights during the training of the feed-forward network to approximate the test function.

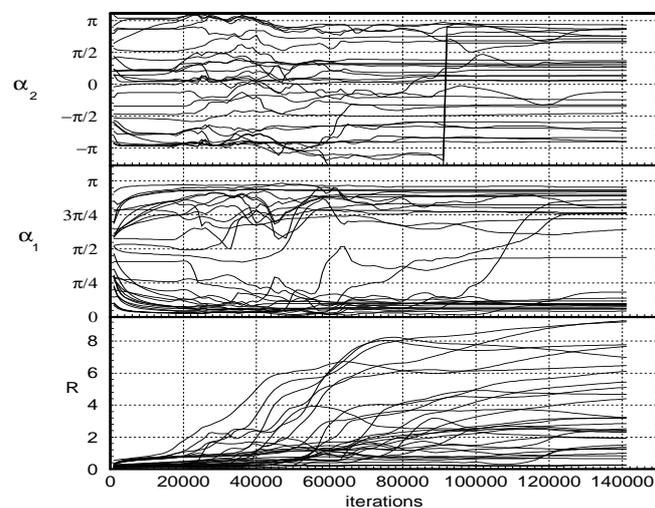


Figure 10.12 Weight dynamics of 30 hidden nodes in learning the 'two spirals' problem

The learning dynamics of weights for problem (10.11) is given in Figure 10.11.

A general picture of weight evolution is more complicated than in the single-node case, as expected. There are 3 groups of hidden units with a different rate of dynamics

- *slow* rate (hidden unit 3);
- *medium* rate (units 1,2,3);
- *fast* rate (unit 5).

A general feature of weight dynamics is that the angular parameters $\alpha_1^{(i)}$ and $\alpha_2^{(i)}$ stabilize earlier than the radius parameter $R^{(i)}$

- after 100 and 1000 iterations for the ‘fast’ unit;
- after 1000 and 2000 iterations for the ‘medium’ units;
- after 2000 and 3000 iterations for the ‘slow’ unit.

The pictures of weight dynamics of the feed-forward network show the properties of the error surface that resemble the case with a single node: at first the angular parameters are optimized, afterwards the optimization is performed along a narrow hyper-ravine when, in essence, only the radius parameter grows.

However, there are cases when the error surface of a feed-forward network is more complicated and very different from the single hidden-node case. There is a well-known example, a “two spirals” classical benchmark problem [140] which is a very difficult problem for a standard back-propagation procedure. Given the desired output tolerance 0.3 for the feed-forward network consisting of 50 Gaussian hidden units, the back-propagation learning (learning step is 0.05 and momentum term is 0.5) succeeds only after ~ 130000 iterations, while the *conjugate gradient* (CG) technique converges much faster, approximately after 3800 iterations. The dynamics of weights during the back-propagation learning of the ‘two-spiral’ problem is shown in Figure 10.12.

Indeed, the weight dynamics points to a very complicated picture of the error surface. It explains why the standard back-propagation procedure is so slow, only advanced procedures of non-linear optimization are relevant in this situation. The CG yields a speedup factor of 34 against the standard back-propagation procedure.

10.8 WHY LOCAL SEARCH WORKS?

The figures show that some important features of error surfaces of a single node also appear in the feed-forward network consisting of several hidden nodes. In certain situations (see “two spirals” problem [140]), the error surface of the feed-forward network can be very complicated. There is no indication of straight, radially oriented ravines as we had seen in the single node case. Consequently, the standard gradient descent does not perform well. The second order methods seem to be better in this case. However, for most real-world applications (the number of weights in the order of 10^4 and the sample size over 10^4) only stochastic (*per sample*) gradient descent is a feasible method in terms of CPU time and required memory.

To succeed with a large problem using a simple gradient descent (and other methods as well) one should know the basic properties of error surfaces and how to initialize the weights. We see the following reasons explaining a relative success of local search procedures:

- symmetry of the weight space;
- good initialization of weights;
- control of non-linearity;
- managing of narrow ravines.

Symmetry of Weights Space.

The symmetry of weight space is due to permutations and sign flips of weights that do not change the network output (see *equi-output transformations* [15]). As a consequence, the weight space is divided into many identical cones and each of them spans a small part.

Depending on the size of a network, a single cone can occupy a very small part of weight space. The feed-forward network with the κ hidden units arranged into a single hidden layer implies 2^κ multiple solutions. For instance, the feed-forward network with 50 hidden units has about $3.4 \cdot 10^{79}$ equi-output transformations.

A great number of identical areas of the error surface makes the problem very difficult for the conventional “space-covering” techniques of global optimization. It suggests that random initialization of weights (small values are preferable) could be useful while starting a local search.

Initialization of Weights

The traditional method of initialization is to generate random weights within the range $[-c/\sqrt{n}, c/\sqrt{n}]$, where n denotes the number of weights leading to a particular node, and c is, in general, a problem specific constant, usually between 0.1 and 1.

Visualization of error surfaces of a single node supports the concept of small initial weights. The vicinity of the origin represents an area where the steepest descent is directed into a “good” ravine, as usual. We may use a “quasi-linear” solution for initial values, too, as demonstrated in a single node case. Such a solution can be found explicitly when R is very small, in the order of 0.1. We call this approach *data-driven* initialization.

At the next phase, when the weights grow and non-linearity of the network becomes noticeable, the search tends to go along the narrow ravine until the deepest point (the global minimum) is found. Thus, in spite of other unfavorable conditions (multiple minima, flat regions, etc.), the local search can be successful. The advantage of data-driven initialization is that the peculiarities of a data set are taken into account.

There are many methods of applying data-driven initialization to neural networks. Most of them can be considered as some type of clustering methods. Data-driven initialization, as usual, improves the performance of ANN [8], [84], [158].

Non-Linearity Control

A well-known semi-heuristic method applying the local search to global optimization is “smoothing”. The idea is to control the smoothing during the process of local optimization [91]. Thus we reach the global optimum, if the initial smoothing transforms the objective function into a uni-modal one [76] and the steps of local optimization and the related smoothing relaxation are small enough. The initial point of the next local optimization is the result of the previous one. The smoothing idea is used in the path-following methods of global optimization (see [49, 58]).

There are different smoothing techniques. For example, we may control the smoothing of error surfaces (10.5) of ANN by gradually increasing the scale of target t . The small scale mean a strong smoothing. The example is a progressive range expansion [55]. Here non-linearity of the system is controlled by

increasing the target values in small steps so that each step can be treated as a small perturbation of the previous one. For sufficiently small steps there is a trajectory leading to the global minimum. The case of zero target scale (the constant output) is solved simply by setting zero weights, except that of the output bias. Afterwards target scale is increased step-by-step and the network is "retrained" using the optimal weight values obtained at the previous step.

We may also control the network non-linearity with the view of preventing trapping into local minimum. Figures 10.5-10.6 show that these minima appear far away from the origin, especially, when the learning set size is small.

A simple way of preventing the weight growth is to add to error function (10.5) the penalty function defined as the squared sum of all weights

$$S(w) = \sum_{i=0}^K w_i^2, \quad (10.12)$$

and then minimize the sum

$$E' = E + \lambda S(w). \quad (10.13)$$

The term (10.12) penalizes flat regions of the error surface (where R is large) by adding a "weight decay" term to the gradient of (10.5).

Managing Narrow Ravines.

Narrow ravines slows down any local search. If a ravine is deep and the learning step is small, there is a little possibility to escape from the ravine.

This shows how important is initialization of the starting point of network weights dynamics. With a properly chosen initialization the network weights converge fast to a "good" ravine and then proceed towards the minimum along the ravine. A preferable case is radially oriented ravines.

A benefit of narrow ravines is that when a point appears to be surrounded by a narrow ravine, we know that some part of network parameters has a very low salience and can be ignored. Thus, it is not necessary to optimize accurately. For example, *early stopping* may prevent an "over-fitting" of the network without a noticeable deterioration of the performance.

10.8.1 Summary

The factor explaining a relative success of local search in ANN learning is a good initialization. A random initialization is a reasonable start because due to symmetries of the weights the solution can be found in any of multiple identical areas. The initialization with small parameter values is preferable. The neighborhood around the origin represents a smooth transition area where the local search often directs the weight vector into a "good" ravine. The well-known regularization techniques, such as smoothing or weight limiting, helps us avoid false minima later on while using local optimization procedures.

Another important factor is that we do not need to know the optimal weights exactly. For example, an early stopping technique may be used as a tool of preventing over-fitting of the neural network. In terms of this book the described ANN optimization techniques may be regarded as a source of heuristics that may be improved by BHA as well as any other heuristics.

PART IV

DISCRETE OPTIMIZATION

11

BAYESIAN APPROACH TO DISCRETE OPTIMIZATION

11.1 INTRODUCTION

Various heuristics are widely used in discrete optimization. The average results of heuristic optimization can be improved by the randomization and optimization of heuristic parameters using the BHA. Therefore we consider the discrete optimization as the main area of BHA application. The representation of discrete optimization as a multi-stage decision problem is also a convenient way to show how BHA works.

11.2 SEQUENTIAL DECISION PROBLEM OF DISCRETE OPTIMIZATION

Consider discrete optimization as a multi-stage decision problem. At each stage $i = 1, \dots, I$ one has to choose a decision $m \in D_i$, $1 \leq m \leq M_i$. Here D_i is a set of feasible decisions at stage i and M_i is the number of feasible decisions. The set D_i usually depends on all previous decisions, which are represented as a vector $d^i = (d_1, \dots, d_{i-1})$, where d_i is a decision at the stage i . Denote by $d = d^{I+1}$ a sequence of decisions at all I stages. The set D defines all feasible decisions d .

Denote by $v(d)$ a value of the objective function that corresponds to the decision d ¹. Denote by v^* the minimum value of the objective function $v(d)$

$$v^* = \min_{d \in D} v(d). \quad (11.1)$$

Here $D^i \subset D$ is the set of feasible decisions (d_i, \dots, d_I)

11.3 EXACT AND APPROXIMATE METHODS OF DISCRETE OPTIMIZATION

Using the well known Branch-and-Bound (B&B) approach we "cut off" bad decisions (bad branches of the decision tree) instead of choosing the best one. We can do this by defining lower bounds², instead of exact values. We cut off those "branches" of the decision tree, whose lower bounds are higher than the best available solution, referred to as "incumbent". Thus, we reduce the amount of calculations, but the general complexity remains high and thus we do not avoid the exponential time algorithm (see Sub-section 2.2.5). For a description of B&B see [144, 119, 134].

The high complexity of exact techniques is one reason to look for simpler approaches. For example, we can apply randomized techniques where decisions are made with some probability $r_i(m)$. We repeat a randomized decision many times and accept the best decision as a result. If r_i is positive for all feasible decisions, $r_i \geq \epsilon > 0$, then the randomized technique converges to the best value of objective v^* with probability one (see Theorem 3.2.1). In the sequel, when discussing convergence, we shall omit the words "with probability one". Usually the final convergence is very slow, thus we stop far away from the optimal value v^* .

¹Speaking about the discrete problems we, as usual, denote by $v(d)$ the "original" objective function of discrete variables d and by $f(x)$ the "auxiliary" objective function of continuous variables x (see expression (11.30)).

²Here we consider minimization.

11.4 DEFINITION OF RANDOMIZED HEURISTICS

Suppose that there exists a simple function $h_i(m)$, referred to as the "heuristic", that roughly predicts the consequences of decisions m . In some cases it is convenient to normalize the heuristics. We use two normalization conditions: the "positive" one and the "plus-minus-one". If

$$0 < a < b, \quad (11.2)$$

where

$$\min_m h_i(m) = a, \quad \max_m h_i(m) = b, \quad i = 1, \dots, I,$$

then we obtain the positive normalization used in this book with the exception of orthogonal polynomial randomization (see Sub-section 11.5.3) and the randomization of the simulated annealing type (see expression (11.23)). The positive normalization does not restrict the heuristics scale. We merely keep the heuristic positive.

If

$$a = -1, \quad b = 1, \quad (11.3)$$

then we have the symmetric normalization that is convenient using an orthogonal randomization (see Sub-section 11.5.3). However, both these normalizations are not always desirable. For example, using normalization (11.3) in the "twin-decision" case, such as simulated annealing (see expression 11.23), we can only see which of the two decisions is better, but not how much better, if a and b are constant.

Usually the heuristics h_i are defined as priority rules. We prefer the decision m which appears to be best now, at the stage i . We disregard future consequences of the decision m . A sequence of heuristics is denoted by $h = (h_i, i = 1, \dots, I)$, where $h_i = h_i(m), i = 1, \dots, I$.

If we "build" a solution from "scratch", then we apply so-called greedy heuristics [59]. If we improve a given feasible decision, then the permutation heuristics are used. The simplest permutation heuristics would be

$$h_i(m) = -v(m). \quad (11.4)$$

Here the objective $v(m)$ has the minus sign, because we regard a greater value of heuristic as better one when minimizing the objective v . Using the permutation

heuristics the set of decisions D_i at each stage i sometimes consists of only two decisions: one after and one before the permutation, as in the Simulated Annealing case (see Sub-section 11.5.4).

11.5 ALGORITHM OF RANDOMIZED HEURISTICS

We take advantage of expert knowledge by relating the probability $r_i(m)$ of decision m to the heuristics $h_i(m)$. The usual assumption is that the probabilities $r_i(m)$ are proportional to heuristics $h_i(m)$. This assumption means linearity of r_i as a function of h_i . If we wish to make the relation more "general", then we have to consider nonlinear functions $r_i = r(h_i)$, too. Let us define a family of functions $r_i = r(h_i)$ by a fixed number of parameters $x = (x_n, n = 1, \dots, N)$. Then we can write $r_i(m) = r(m, h_i, x)$. Here

$$\sum_{m \in D_i} r_i(m) = 1. \quad (11.5)$$

For example, we implement the algorithm of randomized heuristics including condition (11.5) by the following three steps:

Step 1. Divide the unit interval into M_i parts $U(m), m \in D_i$, where length of each part is equal to $r_i(m)$,

Step 2. Generate a random number $\xi \in [0, 1]$ from a uniform distribution.

Step 3. Choose the decision m , if $\xi \in U(m)$.

Now the problem is to obtain the best expression of probabilities r_i as a function of heuristics h_i .

11.5.1 Polynomial Randomization

An ordinary polynomial is a good representation, if we prefer the probability r_i to be expressed as some monotonic function of positive heuristics h_i (see condition (11.2)).

$$r_i^0(m) = r^0(m, h_i, x) = \sum_{n=0}^{N-1} a_{in} x_n h_i^n(m), \quad (11.6)$$

where

$$\sum_{n=0}^{N-1} x_n = 1, \quad x_n \geq 0, \quad n = 0, \dots, N-1, \quad (11.7)$$

and from condition (11.5)

$$a_{in} = \frac{1}{\sum_{m=1}^{M_i} h_i^n(m)}.$$

The number n in expression (11.6) is regarded as the "degree of greed" of the n -th component. The component $n = 0$ means no greed³, because all feasible decisions are equally desirable. If the number of greed n is large, then we prefer the decisions with the best heuristics. Optimizing x we define a "mixture" of degrees of greed such that provide the most efficient randomized decision procedure. The convergence of polynomial randomization follows from condition 11.2 and Theorem 3.2.1.

11.5.2 "Sharp" Polynomial Randomization

In some cases, for example, the travelling salesman problem (see Section 12.2), only small deviations from the pure greedy heuristics are desirable. In such cases a "sharp" polynomial expression (11.8) of probabilities may better correspond to our preferences as compared with "regular" polynomial expressions (11.13) or (11.6):

$$r_i^0(m) = r^0(m, h_i, x) = \sum_{n=1}^N a_{in} x_n h_i^{nS}(m), \quad (11.8)$$

and

$$a_{in} = \frac{1}{\sum_{m=1}^{M_i} h_i^{nS}(m)},$$

where S is large.

³That is a reason why we start index numbering from zero, instead of the conventional one.

11.5.3 Orthogonal Polynomial Randomization

It is well known that orthogonality of polynomials often helps to represent functions using a minimal number of components. The Legendre polynomial L_n is an example in the one-dimensional case:

$$L_n = L_n(y) = \frac{1}{2^n n!} \frac{d^n ((y^2 - 1)^n)}{dx^n}, n = 0, \dots, N - 1 \quad (11.9)$$

Here y denotes h_i .

The first seven components of polynomial (11.9) are as follows:

$$\begin{aligned} L_0 &= 1, \\ L_1(y) &= y, \\ L_2(y) &= \frac{1}{2}(3y^2 - 1), \\ L_3(y) &= \frac{1}{2}(5y^3 - 3y), \\ L_4(y) &= \frac{1}{8}(35y^4 - 30y^2 + 3), \\ L_5(y) &= \frac{1}{8}(63y^5 - 70y^3 + 15y), \\ L_6(y) &= \frac{1}{16}(231y^6 - 315y^4 + 105y^2 - 5). \end{aligned} \quad (11.10)$$

The orthogonality of polynomials $L_n(y)$ means that

$$\int_{-1}^{+1} L_s(y)L_t(y)dy = 0, \text{ if } s \neq t. \quad (11.11)$$

In this expression the integration is from -1 to $+1$. Thus, we assume symmetric normalization (11.3) defined by conditions: $\min_m h_i(m) = -1$, $\max_m h_i(m) = +1$, $i = 1, \dots, I$.

The variance S_n of the polynomial component n is

$$S_n = \int_{-1}^{+1} L_n^2(y)dy = \frac{1}{2n + 1}. \quad (11.12)$$

If the heuristics h_i is multi-dimensional, then each component can be represented as a separate Legendre polynomial. If one would like to take into account the interdependence of components of the multi-dimensional heuristics then one

ought to consider an orthogonal system of multi-dimensional functions. This is a complicated task and will not be considered in this book.

The probabilities of decisions can be represented as the sum of Legendre polynomials L_n multiplied by control parameters x_n .

$$r_i^0(m) = r^0(m, h_i, x) = \sum_{n=0}^{N-1} a_{in} x_n L_n(h_i), \quad (11.13)$$

where $x \in A$ and

$$A = \left\{ x : \sum_{n=0}^{N-1} x_n = 1, x_n \geq 0, n = 0, \dots, N-1 \right\}, \quad (11.14)$$

and from condition (11.5)

$$a_{in} = \frac{1}{\sum_{m=1}^{M_i} L_n(h_i(m))}. \quad (11.15)$$

We regard the product of the control parameter x_n and variance S_n as a "weight" w_n of the component n :

$$w_n = x_n S_n. \quad (11.16)$$

This definition can be useful when analyzing the results of optimization of the control parameter x .

We need some correction to expressions (11.13)-(11.15), if the convergence condition $r_i^0(m) \geq \epsilon > 0$ does not hold for some $m \in D_i$ and some i . Define the set $D_i(\epsilon)$ as a set of all m such that $r_i^0(m) < \epsilon$. Then the corrected probabilities $r_i(m)$ are as follows:

$$r_i(m) = \begin{cases} \epsilon, & \text{if } m \in D_i(\epsilon) \\ r_i^0(m) \delta_i & \text{otherwise.} \end{cases} \quad (11.17)$$

Here δ_i is the correction multiplier:

$$\delta_i = \sum_{m \notin D_i(\epsilon)} r_i^0(m) + |D_i(\epsilon)| \epsilon \quad (11.18)$$

where $|D|$ is the number of elements of the set D .

We deviate from orthogonality conditions (11.11) using the convergence correction (11.17). Usually this deviation is not significant.

A disadvantage of representation (11.13) is that the probability r_i is not a monotonic function of heuristics. One can understand the results easier, if the components of the expression r_i are monotonous.

11.5.4 Non-smooth Randomization

Delta Randomization

We considered two polynomial randomizations: one orthogonal, similar to the Legendre polynomial, and one non-orthogonal, similar to the Taylor expansion. It would be convenient to extend the latter by adding a "Delta" term representing "pure" heuristics

$$r_i(m) = \sum_{n=0}^{N-1} a_{in} x_n h_i^n(m) + x_N \Delta_i(m), \quad (11.19)$$

where the Delta term

$$\Delta_i(m) = \begin{cases} 1, & \text{if } h_i(m) = \max_{k \in \{1, \dots, M\}} h_i(k), \\ 0, & \text{otherwise.} \end{cases} \quad (11.20)$$

Here h_i satisfies condition (11.2) and x_n satisfies condition (11.7).

From condition (11.5)

$$a_{in} = \frac{1}{\sum_{m=1}^{M_i} h_i^n(m)}.$$

To insure the convergence of Delta randomization we limit the probability of the Delta term:

$$x_N \leq 1 - \epsilon, \quad \epsilon > 0. \quad (11.21)$$

Then the convergence follows from condition (11.2) and Theorem 3.2.1.

The pure heuristics term (11.21) may be considered as a limit on the last term in polynomial expressions (11.6) and (11.8).

Consider, as an illustration, the "twin-case" where $m \in \{0, 1\}$, $M = 2$, and $N = 1$. Here $m = 0$ denotes remaining in the "old" state, and $m = 1$ defines going to the "new" state that we obtain after the permutation operation. In this special case we optimize only one independent parameter x_0 ⁴ defining the probability of using Monte Carlo randomization $r_i(0) = r_i(1) = 1/2$. Thus, we are looking for the optimal "lottery" of simple Monte Carlo and "pure" heuristics. This way we define an alternative to the randomization of the simulated annealing type (see the next subsection).

⁴The parameter defining the probability of using pure heuristics $x_1 = 1 - x_0$.

In the "triplet-case" where $N = 2$ one considers three terms: x_0 defining the probability of using Monte Carlo randomization, x_1 defining the probability of linear randomization, and x_2 defining that of pure heuristics. This Delta-triplet could be better in many cases as compared with the Taylor-triplet: constant, linear, and quadratic terms.

Simulated Annealing

A popular method of global optimization, namely, the simulated annealing, may be considered in the framework of the Bayesian heuristics approach. We use the following heuristics⁵, convenient for simulated annealing.

$$h_i(m) = -(v(m) - v(\bar{m})), \quad m = 0, 1. \quad (11.22)$$

Here

$v(m)$ is the objective after the permutation i ,

$v(\bar{m})$ is the objective before the permutation i ,

$D_i = \{m, \bar{m}\}$, which means that the number of decisions $|M_i| = 2$.

The randomized permutation heuristics is:

$$r_i(m) = \begin{cases} e^{\frac{h(m)}{x/\ln(1+i)}}, & \text{if } h(m) < 0 \\ 1, & \text{otherwise,} \end{cases} \quad (11.23)$$

where

i is the iteration number,

x is the "initial temperature".

The difference from the usual simulating annealing is that we optimize the parameter x for some fixed number of iterations. We disregard the asymptotic behavior because the asymptotic properties are beyond the scope of the Bayesian Heuristics Approach.

GRASP System

Consider in the framework of the BHA the well known system GRASP (see [42, 118, 130, 116, 117, 131]) of optimization using heuristics.

⁵Note that these heuristics do not satisfy the normalization conditions (11.3) and (11.2).

GRASP Algorithm

The essence of the GRASP algorithm (see [129]) is shown in Figure 11.1 The

```

cost_best=INFINITY;
for (k=1 to MAX_TRIES)
{
    s = construct_greedy_randomized_solution();
    s = local_search(s);
    if (c(s) < cost_best)
    {
        s_best = s;
        cost_best=c(s);
    }
}

```

Figure 11.1 GRASP Nutshell

heuristic is repeated MAX_TRIES times. During each iteration a greedy randomized solution is constructed and the neighborhood around that solution is searched for a local optimum. The component "construct_greedy_randomized_solution()" constructs a solution, one element at a time (in the style of a greedy heuristic), by doing the following:

1. apply to all not yet unselected candidates m a GREEDY function $h(m)$;
2. sort these candidates according to the GREEDY function;
3. select a subset, consisting of GOOD (but not necessarily BEST) candidates to be in Restricted Candidate List (RCL);
4. select a CANDIDATE m , at RANDOM with probability r_m , from RCL to be in the solution;
5. change GREEDY function to take into account the inclusion of the candidate m into the solution.

These steps are repeated until a solution is constructed.

BHA Version of GRASP

In this paragraph we describe how to apply BHA to implement a GRASP type algorithm. We will outline the differences and additions to the conventional GRASP version (see Figure 11.1) and explain them

1. It is supposed that using BHA the convergence conditions of Theorem 3.2.1 should be satisfied. This is a difficult task, if the candidate list is restricted (see RCL area in GRASP). Thus applying BHA in GRASP one should provide positive probabilities

$$r_m \geq \epsilon, \epsilon > 0, \text{ for all } m.$$

2. Selecting a candidate from RCL by BHA not merely one but the entire "arsenal" of different heuristics and randomization rules might be applied.
3. Some "lottery" parameters $x = (x_1, \dots, x_n)$ defining the "mixture" of different randomization procedures and/or different heuristics are determined by multiple repetitions using Bayesian methods of continuous global stochastic optimization.

For example, the GRASP heuristic for the set covering problem [42, 22] may be regarded in BHA terms as a step-function randomization

$$r(m) = \begin{cases} r_0, & \text{if } |P_m| \geq \alpha \max_{1 \leq j \leq J} |P_j| \\ 0, & \text{otherwise.} \end{cases} \quad (11.24)$$

Here $\alpha < 1$ is a threshold factor defining the RCL set and $|P_j|$ is the cardinality of set P_j (see [22]), and J is the number of available candidates.

An example of a simple BHA implementation of GRASP heuristics would be a mixture $x = (x_1, x_2)$ of two different randomizations: the traditional RCL defined by expression (11.24) and the linear one, defined by the following expression

$$r_i(m) = \frac{|P_m|}{\sum_{j=1}^i |P_j|}. \quad (11.25)$$

In the GRASP-BHA case, the parameter $x_1 \in [0, 1]$ defines the probability of applying randomization (11.24) and the parameter x_2 defines that of randomization (11.25). The vector $x = (x_1, x_2)$ where $x_2 = 1 - x_1$, $x_1 \in [0, 1 - \epsilon]$

is optimized by solving an one-dimensional continuous global stochastic optimization problem applying the Bayesian methods and using the best results of multiple repetition as an objective function.

The theoretical advantage is that this GRASP-BHA version satisfies the conditions of Theorem 3.2.1⁶ thus assuring the convergence in the sense of conditions (3.11) and (3.5). Numerous other GRASP-BHA versions could be designed. We considered this one merely as an illustration because BHA is a conceptual framework which produce good results if applied by an expert in the specific heuristics.

11.5.5 Transformation to a Unit Box

The equality constraint (11.14) is not convenient for global optimization if $N > 2$. Therefore we transform the set of feasible values of x , defined by condition (11.14), to a "unit box":

$$A^0 = \{x^0 : 0 \leq x_n^0 \leq 1, n = 0, \dots, N-1\}. \quad (11.26)$$

We do that by a simple transformation

$$x_n = \frac{x_n^0}{\sum_{n=0}^{N-1} x_n^0} \quad n = 1, \dots, N-1. \quad (11.27)$$

Expression (11.27) transforms the objective function $f(x)$, defined on the set A into some other function $f^0(x^0) = f(x^0(x))$ defined on the N - dimensional unit cube A^0 (see conditions(11.26)). The notation $x^0(x)$ means that x^0 depends on x .

Let us fix some $x \in A$. Then the values of the function $f_0(x^0)$, remain equal to the value $f(x^0(x))$ for all x^0 satisfying condition (11.27) for fixed x .

It follows from condition (11.27) that for fixed $x = (x_0, \dots, x_{N-1})$

$$x_n^0 = \frac{x_n}{x_0} x_0^0, \quad n = 1, \dots, N-1.$$

Fixing $x_0^0 = 0$, we obtain the zero point $x_n^0 = 0, n = 1, \dots, N-1$. Fixing $x_0^0 = x_0$, we obtain the point $x_n^0 = x_n, n = 1, \dots, N-1$. This means that

⁶ Assuming that both $|P_m|$ and x_2 are positive.

$x^0 = x$. Thus, expression (11.27) defines the line in the set A^0 going from zero through the point $x^0 = x$.

This property of the function $f^0(x^0)$ defined by transformation (11.27) is usually not very important in applying global optimization methods. It does not contradict the basic assumptions of Bayesian methods designed for the global optimization in the unit "box" (see [100]). The peculiarity of the function $f^0(x^0)$, related to transformation (11.27), seems to be less important in comparison with computational advantages of the unit box construction.

11.6 REDUCTION TO CONTINUOUS STOCHASTIC OPTIMIZATION

Denote by $v(K, x)$ the expectation of the best value of the objective function, if the randomized decision $r(x)$ is repeated K times for the fixed parameter x . We assume that $r(x)$ is such that the probability of any feasible decision is positive. In such a case the best value of the objective function converges to the exact optimum v^* , if the number of repetitions K is large (see Theorem 3.2.1),

$$\lim_{K \rightarrow \infty} v(K, x) = v^*, \quad x \in A. \quad (11.28)$$

The exact computation of $v(K, x)$ is no less complicated than the solution of the original discrete optimization problem (11.1). A natural and convenient way of estimating $v(K, x)$ is by Monte Carlo simulation. Let $f_K(x)$ be the best value of the objective function obtained after K repetitions of the randomized decision procedure $r(x)$ for fixed x . The expectation of $f_K(x)$ is equal to $v(K, x)$ by definition. We defined the randomized decision procedure so that

$$\lim_{K \rightarrow \infty} f_K(x) = v^*, \quad x \in A. \quad (11.29)$$

This means that all the values of parameters x are "good" asymptotically. However the right choice of x could be very important, if the number of repetitions K is not very large.

Thus we obtain the following continuous problem of stochastic programming

$$\min_x f(x), \quad (11.30)$$

where

$$f(x) = f_K(x),$$

and

$$x = (x_0, \dots, x_{N-1}), \quad \sum_{n=0}^{N-1} x_n = 1, \quad x_i \geq 0, \quad n = 0, \dots, N-1. \quad (11.31)$$

Thus we reduced, in the described sense, discrete optimization problem (11.1) to the problem of continuous stochastic optimization (11.30) (11.31).

11.7 CONVERGENCE

Let R be the number of iterations at each repetition of the stochastic optimization. Thus, the total number of observations K_T (calculations of the objective function $v(d)$ for the fixed decision d) is the product of iterations R and repetitions K , therefore $K_T = RK$. Replace condition (11.29) by a weaker condition

$$\lim_{K_T \rightarrow \infty} f_K(x) = v^*, \quad x \in A, \quad \text{and } K \geq 1. \quad (11.32)$$

This expression follows directly from the convergence condition $r_i^0(m) \geq \epsilon > 0, n = 1, \dots, M_i, i = 1, \dots, I$ and Theorem 3.2.1.

Condition (11.32) means that the method will achieve the global optimum for any fixed number of repetitions $K \geq 1$, if the total number of observations $K_T = KR$ is large enough. However, the average results of optimization for the fixed number K_T may depend significantly on the number of repetitions K . Therefore, for any specific problem some preliminary investigation is desirable to define the right number of repetitions K .

11.8 IMPROVING EFFICIENCY

We expect the algorithm employing randomized heuristics to be efficient, if v^* statistically depends on some simple heuristic h_i . Thus, in the first stage we investigate the "quality" of heuristics using the Delta-triplet (see (11.19) and (11.20)).

The heuristics h_i is considered to be useless, if the optimal value of the zero component x_0 , obtained using three terms of expressions (11.19) and (11.20), is near to unit. The reason is that we have obtained the best result using the simplest uniform Monte Carlo method, without any heuristics.

The heuristics h_i is regarded to be very good, if the optimal value of the second component x_2 is near to unit. Here we obtained the best result by choosing a decision that corresponds to the best heuristics, without any randomization and parameterization.

The heuristics h_i could be improved by randomization, if the optimal value of the component x_1 was significantly greater than zero. In this case, we proceed to the second stage of optimization. In the second stage, we use the randomized procedure $r(x)$ with probabilities defined by expression (11.13), or by (11.17) or (11.6). The optimization of parameters x should make the search more efficient.

If we see that the optimization of parameters x improves the results significantly, then we proceed to the third stage. In the third stage, we repeat the randomized decision procedure $r(x)$ for fixed values of x which had been obtained in the second stage of optimization. Of course, depending on the specific conditions of the problem, some optimization steps may be omitted.

The number of repetitions K may be different in different stages, namely, K_1, K_2 , and K_3 . The total number of observations here is

$$K_T = K_1 R_1 + K_2 R_2 + K_3,$$

where R_1 is the number of iterations in the first stage of optimization, and R_2 is the number of iterations in the second stage.

The right choice of repetition numbers K_1, K_2 , and K_3 depends on several factors. The "cost" of one observation is an important factor. We set the repetition numbers $K_1 = K_2 = 1$, and $K_3 = 0$, if the cost of an observation is higher than that of the auxiliary computations. We use the auxiliary computations to define the next value of x at each iteration of optimization. Bayesian methods need a large number of such computations to minimize risk functions (4.9) and (4.11).

If observations are "cheap", then one should set larger numbers of repetitions K_1, K_2 . The aim is to make the cost of observations at each iteration close to the cost of auxiliary computations. Since the auxiliary computations are defined by the optimization method, one cannot control them. However, we can increase the cost of observations in one iteration by increasing the number of repetitions. The results are improved by setting a large number of "post-optimization" repetitions K_3 , if observations are cheap.

The described methods are most important in solving families of related discrete optimization problems. Here one may optimize x for only one typical problem of the family applying the optimal values of x to many related problems. In this way, we can reduce the optimization "cost" by dividing it between many problems. In the examples of this book we optimize parameters x only for individual problems. Thus we show only a part of the advantages of the Bayesian approach. Section 3.4 and Figures 12.1 and 12.2 illustrate the "learning" potential of the Bayesian heuristics approach, when the optimal x of one problem is extended to the whole family of related problems.

11.9 METHODS OF GLOBAL STOCHASTIC OPTIMIZATION

There are several specific features of problem (11.30) (11.31) that make it difficult for optimization:

1. the function $f_K(x)$ is stochastic;
2. the function $f_K(x)$ is multi-modal (usually);
3. the function $f_K(x)$ is expensive (one observation requires many calculations).

There exist techniques of stochastic approximation (see [39]) that converge to local minimum with probability one. There are methods of global stochastic approximation (see [152]) that provide a convergence to the global minimum. Thus, both the first and the second conditions are met by traditional methods.

The convergence of conventional methods of global stochastic optimization, such as global stochastic approximation or simulated annealing, is usually very slow (see [168]). Sometimes the convergence condition appears almost irrelevant, if the number of iterations is not very large. The reason for such a behavior is that these methods are designed for the optimization of inexpensive functions. If the function is inexpensive, then the number of iterations is usually large and the asymptotic performance is important.

Therefore, to meet all three conditions one needs a search technique that might minimize the expected deviation from the global minimum, when the number

of iterations (evaluations of the function $f_K(x)$) is small. We define such techniques as Bayesian methods of global optimization (see [100]). We apply the Bayesian approach to discrete optimization using the Bayesian optimization for continuous auxiliary problem (11.30) (11.31). However, "non-Bayesian" methods of global stochastic optimization could also be used for this problem. We regard the BHA as an important special case of the Adaptive Heuristic Approach (AHA), where heuristics are adapted to the problems.

The main advantage of the Bayesian Heuristics Approach to the discrete optimization is that it shows how to involve expert knowledge about the problem into the general mathematical framework. We include the expert knowledge directly into heuristics h_i ⁷. The mathematical part refers to the auxiliary optimization problem (11.30) (11.31). If the choice of heuristics h_i is good, then the optimization of auxiliary continuous problem (11.30) (11.31) helps to solve the original discrete problem (11.1) more efficiently. If not, then the results of optimization of problem (11.30) (11.31) indicate that the heuristics h_i is irrelevant to original discrete optimization problem (11.1).

11.10 APPLICATION TO STOCHASTIC DISCRETE OPTIMIZATION

Suppose that the objective v depends on a random parameter θ . Denote by $v(K, x, \theta_l)$ the expectation of the best value of the objective function if the randomized decision $r(x)$ is repeated K times. Here the value of the control parameter is x and the value of the random parameter θ is θ_l , where $l = 1, \dots, L$. Denote by $q = (q_l, l = 1, \dots, L)$ a probability distribution of the parameter θ . We define as K the number of optimizing repetitions.

Denote by $v(K, x)$ the expected value of the function $v(K, x, \theta)$ with respect to the probability distribution q of the random parameter θ .

Denote by $f_K(x, l)$ the best value of the objective function obtained after K repetitions of the randomized decision $r(x)$. We fix the control parameter x and the random parameter $\theta = \theta_l$. We repeat this procedure $K(l)$ times for each $l = 1, \dots, L$ and define $K(l)$ as the number of averaging repetitions. The average

⁷We include the expert knowledge indirectly, too, while defining the a priori distribution on a set of heuristics randomization parameters.

of the stochastic function $f_K(x, l)$ is defined by the following expression:

$$f_K(x) = \frac{1}{L} \sum_{l=1}^L \frac{1}{K(l)} \sum_{k=1}^{K(l)} f_K(x, l). \quad (11.33)$$

It is not difficult to show that the expectation of $f_K(x)$ is equal to $v(K, x)$ if

$$K(l) = q_l K_0. \quad (11.34)$$

Here K_0 is the number of averaging repetitions corresponding to the minimal probability $\min_{1 \leq l \leq L} q_l$. The number K_0 should not be small. This means that the stochastic discrete optimization needs many of repetitions.

We optimize randomized decision procedures r_i by the same expression (11.30) (11.31) as in the deterministic case. That is an important advantage of BHA.

12

EXAMPLES OF DISCRETE OPTIMIZATION

12.1 KNAPSACK PROBLEM

12.1.1 Definition of Heuristics

The knapsack problem (see expression (16.61)) is to maximize the total value of a collection of objects when the total weight g of those objects is limited. We denote the value of the object i by c_i and the weight by g_i .

The decision m means that object m is selected.

Define the heuristics h_i satisfying expression (11.2)

$$h_i(m) = \frac{a_m - A_i}{A^i - A_i} + a, \quad (12.1)$$

where $a_m = c_m/g_m$, and

$$A_i = \min_{m \in D_i} a_m, \quad A^i = \max_{m \in D_i} a_m, \quad a > 0. \quad (12.2)$$

Here

$$m \in D_i, \quad \text{iff} \quad \sum_{k=1}^i g_k \leq g. \quad (12.3)$$

Sum (12.3) denotes the total weight of objects taken during the sequence of i decisions.

Expression (12.1) defines the well known and widely used "greedy" heuristics.

Define the randomized procedure $r(x)$ by expression (11.6)

$$r_i = r(h_i, x) = \sum_{n=0}^{N-1} a_{in} x_n h_i^n(m). \quad (12.4)$$

The number n denotes the "degree of greed" of n -th component. The number $n = 0$ means no greed component, because all feasible decisions are equally desirable.

Monotonic expression (11.6) of r_i is used in this and the following examples. Orthogonal expression (11.13) or non-smooth expression (11.19)(11.20) may be used as well, depending on the situation.

One optimizes the randomized decision function (12.4) by solving stochastic optimization problem (11.30) (11.31) where function $f_K(x)$ defines the maximal feasible sum of weights. We apply the Bayesian methods (see [100, 96, 102]) for the optimization of $f_K(x)$. The solutions to the knapsack problem using various methods are demonstrated in the form of four tables and related comments. The results of the Bayesian, deterministic heuristics, and exact B&B method are compared.

The average results were obtained by repeating the optimization procedures for 100 times with the same data, the same probability distributions, and different random sequences. Table 12.1 illustrates the results of the Bayesian algorithm

Table 12.1 Comparison of the Bayesian method and the exact one

$K_B = 100$, and $K = 1$							
N_O	K_E	f_B	f_E	$\delta_B \%$	$x_B(0)$	$x_B(1)$	$x_B(2)$
50	313	9.56057	9.62347	0.654	0.0114	0.0280	0.9605
100	526	13.0703	13.1241	0.411	0.0316	0.0412	0.9271
150	771	16.6301	16.6301	0.000	0.0150	0.1945	0.7904
200	875	37.4665	37.4859	0.050	0.0315	0.0530	0.9437
250	568	53.7781	53.9000	0.226	0.0091	0.0511	0.9397
300	1073	28.3144	28.6106	1.034	0.0113	0.0835	0.9050
350	1416	30.4016	31.7527	4.254	0.0064	0.0646	0.9288
400	2876	32.1632	33.3192	3.469	0.0202	0.0452	0.9344
450	1038	105.467	105.578	0.105	0.0101	0.0149	0.9748
500	2132	39.3583	42.1047	6.521	0.0078	0.1556	0.8365

and that of the exact method. In this table the symbol N_0 stands for the number of objects, K stands for the number of repetitions, K_B denotes the total number of observations by the Bayesian method, and K_E denotes the number of nodes considered by the exact method. The symbol f_B denotes the best result of the Bayesian method, f_E denotes the exact optimum, $\delta_B\%$ denotes the mean percentage error of the Bayesian algorithm, and $x_B(n)$, $n = 0, 1, 2$ denote the optimal values obtained by the Bayesian method.

If the deviation of some solution does not exceed 5%, we refer to it as a 5% solution. One can expect that the 5% solution satisfies the applications, where the level of data uncertainty is not less than 5%. It is not an unusual level in many applications.

Table 12.1 shows that we need to consider from 313 to 2876 nodes to obtain the exact solution, while only 100 observations are needed to obtain the 5% solution by the Bayesian method. The deviation exceeds 1% only for three cases in ten. The average deviation is 1.67%.

Assume that roughly the same computing time is necessary for one node and for one observation. As seen from Table 12.1, under this assumption the Bayesian 5% solution is about three times "cheaper" as compared to the exact one, if the number of objects is 50. If this number is 400, then the Bayesian 5% solution is almost 30 times cheaper.

Table 12.2 demonstrates the results of three different methods. In Table 12.2, the symbol f_H denotes the best result of the deterministic heuristics method. Comparing Table 12.2 with Table 12.1 we see the improvement of results when the number of Bayesian observations is increased from 100 to 1000. For 1000 Bayesian observations, the deviation is less than 1% in all the ten cases. So we need a large number of observations, if we wish to obtain a higher than 1% accuracy by the Bayesian methods.

Table 12.1 and Table 12.2 suggest that the most efficient method seems to be close to the deterministic heuristics, if we accept the 1% accuracy. If we wish a higher accuracy, then we have to choose: either to use the exact method, or to increase the number of Bayesian observations.

Table 12.3 shows that the exact methods can be more economical, if we need the average accuracy to be much higher than 1%. Table 12.3 illustrates that using the Bayesian approach one can reach the exact optimum with an accuracy of 0.1% by increasing the number of observations up to 10,000 (100 repetitions

Table 12.2 Comparison of the Bayesian, the deterministic heuristics, and the exact methods

The case when $K_B = 1000$ and $K = 1$							
N_O	K_E	f_B	f_H	f_E	$x_B(0)$	$x_B(1)$	$x_B(2)$
50	313	9.6234	9.5605	9.6234	0.0569	0.2722	0.6708
100	371	15.9162	15.7361	16.1537	0.1174	0.0075	0.8750
150	554	22.7691	22.6892	22.7691	0.0820	0.1537	0.7641
200	1119	12.5672	12.5985	12.7005	0.0412	0.3989	0.5597
250	645	60.2354	60.2058	60.3298	0.0438	0.0267	0.9294
300	2288	40.3135	40.8855	41.2118	0.0197	0.0227	0.9574
350	908	79.5732	79.5914	79.6651	0.0481	0.0801	0.8717
400	1745	36.2534	36.5369	36.5394	0.0101	0.0686	0.9212
450	895	108.022	108.019	108.034	0.0348	0.0329	0.9322
500	1901	96.1148	96.4902	96.1148	0.0031	0.0097	0.8997

Table 12.3 Comparison of the average results of the Bayesian and the exact method

$K_B = 10000,$ $R = 100, K = 100, \text{ and } K_A = 100$				
N_O	K_E	f_B	f_E	$\delta_B\%$
50	313	9.623	9.623	0.000
100	311	9.705	9.704	0.010
500	1340	44.753	44.717	0.080

multiplied by 100 iterations). However, we may obtain the exact optimum much cheaper by the exact B&B techniques.

We can roughly estimate the relation between the average deviation and the number of observations as an inverse square root. For instance, one can expect to decrease the deviation ten times by increasing the observation number a hundred times. The deviation also depends on many other features, including the relation between the number of repetitions K and the number of iterations K_B .

We did mention in Section 11.9 that Bayesian methods are not only ones for stochastic global optimization problem (11.30) (11.31). The theoretical reason of using Bayesian methods is the possibility to include specific heuristics into a general Bayesian framework. Table 12.4 demonstrates results of using a simple Monte-Carlo technique to solve stochastic problem (11.30) (11.31) instead of sophisticated Bayesian methods. By comparing Table 12.1 and Table 12.4 we see that 10 times more Monte-Carlo iterations are necessary to approach the accuracy of the Bayesian method. The number of Bayesian iterations is $K_B = 100$, and the number of Monte Carlo iterations is $K_M = 1000$. Average deviations are about 1.3% and 1.5%, respectively. In Table 12.4 symbols

Table 12.4 Comparison of the Monte Carlo method and the exact one

$K_M = 1000$ and $K = 1$							
N_O	K_E	f_M	f_E	$\delta_M\%$	$x_M(0)$	$x_M(1)$	$x_M(2)$
50	313	9.623	9.623	0.000	0.0569	0.2722	0.6708
100	502	7.990	7.990	0.000	0.1174	0.0075	0.8750
150	1059	10.898	10.958	0.547	0.0820	0.1537	0.7641
200	105	54.210	54.212	0.003	0.0412	0.3989	0.5597
250	868	57.127	57.184	0.099	0.0438	0.0267	0.9294
300	1377	35.233	36.019	2.182	0.0197	0.0227	0.9574
350	1885	26.834	30.051	10.705	0.0481	0.0801	0.8717
400	1356	89.948	90.327	0.419	0.0101	0.0686	0.9212
450	1800	82.286	83.430	1.371	0.0348	0.0329	0.9322
500	1090	93.988	94.903	0.964	0.0031	0.1970	0.8997

f_M , $x_M(0)$, $x_M(1)$, $x_M(2)$ denote the results of optimization of the parameters x by the Monte Carlo method using a uniform distribution. The number of Monte Carlo iterations is K_M .

An unexpected result of the knapsack example is the conclusion that the best approximate method seems to be the pure heuristics if about 1% accuracy is needed. Consequently in such a case no optimization of parameters x is necessary. If we wish a higher accuracy, then we have to choose: either to use the exact method, or to increase the number of Bayesian observations. The exact methods are more economical, if we need the average accuracy to be about 0.1%.

We see that applying BHA to knapsack problem we did not obtain significant positive results. Another examples in this book, in particular applying BHA to a family of scheduling problems (see Section 12.3 and Chapters 14 and 15), demonstrate efficiency of BHA.

12.2 TRAVELLING SALESMAN PROBLEM

The problem is to minimize the total path length of a salesman visiting I cities and returning home (see[92]). The decision $m \in D_i$ means to go at the stage i to the city number $m \in D_i$. Here the set D_i is a set of cities the salesman has not visited in the previous $i - 1$ stages.

The number c_m denotes the length of the path between the city m and the last city, he has visited in the $i - 1$ previous stages.

Consider the greedy heuristics satisfying expression (11.2)

$$h_i(m) = \frac{C^i - c_m}{C^i - C_i} + a, \quad (12.5)$$

where

$$C_i = \min_{d_i(m) \in D_i} c_m, \quad C^i = \max_{d_i(m) \in D_i} c_m, \quad a > 0. \quad (12.6)$$

Here c_m is with minus sign, because in this book greater heuristics are regarded as the better ones. It is well known that going to distant cities one usually misses the optimal decision. However some deviation from the pure greedy decision (going to the nearest city) may be justified. Therefore we define the randomized decision function r_i by the "sharp" polynomial expression (11.8), where the number of terms $N = 3$.

$$r_i^0(m) = r^0(h_i, x) = \sum_{n=1}^3 a_{in} x_n h_i^{nS}, \quad (12.7)$$

where $r_i^0(m)$ determines the probability of going to city m .

We used numbers S of order 100. In this case, the probability of going to some distant city is almost zero. However the probability of some small deviation from the pure heuristics is significant. For example, the probability of going

not to the nearest city (in a problem of 100 cities) is about 1% in one iteration and about 27% in 100 iterations. One may consider these sparse "deviations" from the pure heuristic decisions as some new "initial" points preventing the heuristics to be trapped and thus providing the convergence.

We optimize randomized decision function (12.7) by solving a stochastic optimization problem (11.30) (11.31) where $f_K(x)$ denotes the shortest path found as a result of K repetitions.

In numerical experiments "cities" were considered to be points in the ten-dimensional unit cube. The multi-dimensional case is less realistic, but more convenient for comparison of different methods. The reason is that the choice of alternative paths is wider here.

The distances between the cities are defined as Euclidean distances between the corresponding points. The number of cities I is from 100 to 500. I points (representing cities) are generated 300 times, by sampling from a uniform distribution in the 10-dimensional unit cube.

Usually the exact optimum is not obtained, because the number of problems (5×300) and the size of the problems (from 100 to 500 cities) is too large. Consequently we merely compare the Bayesian methods with the pure heuristics. Both a simple sequential nearest city algorithm and a more complicated local permutation algorithm are considered.

We execute the randomized algorithm (11.6) for each vector of parameters $x = (x_1, x_2, x_3)$ defined by the Bayesian global optimization method (see [100]). A length of the shortest path is the result.

For purposes of the algorithm involving local permutations, some initial travel path must be used. We try to improve this initial fixed path by selecting a pair of adjacent cities $A - B$ and afterwards considering another pair of adjacent cities $C - D$. The pairs are chosen so that reconnecting A to C and B to D we still obtain a path that visits all cities. We seek such a new path that is shorter. The initial path is selected by a greedy heuristic, then we repeat I times the following two steps:

- choose the first pair of cities at random;
- consider the remaining pairs in succession as long as a shorter path is found.

The local permutation heuristics h_m is defined by the same expressions (12.5) and (12.6) as in the case of greedy heuristics. However, the parameter c_m is different and determines the total length of travelling salesman path corresponding to the permutation m .

Table 12.5 illustrates the results of Bayesian algorithms employing simple greedy heuristics (11.6). The results of pure greedy heuristics and that of pure local permutation are shown, too, for comparison. In this table the letter B stands for

Table 12.5 Results of the Bayesian method using greedy heuristics.

I	S	f_G	f_B	f_{GL}	f_{G-B}	d_{G-B}
100	64	78.90	77.62	77	1.28	0.39
200	128	144.88	143.30	142	1.58	0.71
300	128	205.90	203.95	202	1.95	0.85
400	128	264.62	262.63	260	1.99	1.05
500	128	321.35	319.10	316	2.25	1.32

the Bayesian method, the letter G denotes pure greedy heuristics, and the symbol GL denotes pure local permutations. The table provides sample means f and sample variances d . Thus f_{G-B} denotes the mean of the difference between the results of a greedy heuristic and the Bayesian methods, and d_{G-B} denotes the corresponding variance. The symbol I stands for the number of cities, and the letter S is the power of the "sharp" polynomial (see expression (12.7)). The Bayesian method performs 46 observations.

The results show that the Bayesian method is roughly 1% better than the pure greedy heuristic. The improvement declines with the size of the problem, from 1.6% for $I = 100$ to 0.7% for $I = 500$. Comparing the columns f_{GL} and f_{B-GL} we see the advantage of local permutation heuristics. Therefore, the Bayesian method is applied here, too. It is implemented in the following algorithm.

1. Select an initial path joining all the cities.
2. Select a pair $A - B$ of cities from all the pairs of adjacent cities.
3. Define the heuristic h on the remaining pairs of adjacent cities $C - D$ ($A \neq C, A \neq D, B \neq C$) so that h is proportional to the quantity $\text{dist}(A, B) + \text{dist}(C, D) - \text{dist}(A, C) - \text{dist}(B, D)$ and normalized as in equation (12.5).

4. Select an edge $C - D$ based on the heuristic h and the parameters \mathbf{x} as in equations (12.7).
5. If the stopping condition is not satisfied, repeat steps 2-5.

Table 12.6 demonstrates the results of Bayesian algorithms for the case of local permutation heuristics. The results of pure greedy heuristics and that of pure local permutation are shown, too. In Table 12.6 the symbol BL stands for the

Table 12.6 Results of the Bayesian method using permutation heuristics.

I	S	f_G	f_{GL}	f_{BL}	f_{G-BL}	d_{G-BL}	f_{GL-BL}
100	32	78.7	77	76	2.7	0.6	1
200	32	144.5	142	140	4.5	0.7	2
300	32	206.1	202	200	6.1	1.2	2
400	32	265.3	260	258	7.3	1.4	2
500	32	321.5	316	313	8.5	1.5	3

Bayesian method, the symbol GL denotes a pure local permutation heuristic. The table provides sample means f and sample variances d . Thus, the symbol f_{GL-BL} denotes the average difference between the pure local permutation heuristic and the Bayesian method. The symbol f_{G-BL} denotes the average difference between greedy heuristics and the Bayesian method using local permutations. The symbol d_{G-BL} denotes the corresponding variance. Here the Bayesian method performed 26 observations and the algorithm stops after 50 repetitions.

Sharp polynomial randomization (11.8) was chosen as a result of some additional experimentation. When the uniform distribution was included (as in expression (11.6)), the average value of f_{G-BL} was 1.5 for $I = 100$ (one hundred cities). Using expression (11.8) with $S = 1$, the average value of f_{G-BL} was 2.5 for the same $I = 100$. The best average gain was achieved with $S = 32$ (see Table 12.6). The distribution of coefficients x_i from expression (11.8) or expression (11.6) shows which powers of h_i are most important in calculating $r_i^0(m)$.

The results of Table 12.6 show that the Bayesian method is approximately 1% better than the pure heuristics (see the average gain f_{GL-BL}). The improvement is almost independent of the size of the problem. We consider 1% to be a good result, because this improvement is obtained near the global minimum, where even a fraction of percent is important.

12.3 FLOW-SHOP PROBLEM

12.3.1 Definition

We denote by J and S the set of jobs and machines. Denote by $\tau_{j,s}$ the duration of operation (j, s) , where $j \in J$ denotes a job and $s \in S$ denotes a machine. Assumption $\tau_{j,s} = 0$ means that operation (j, s) is irrelevant.

Suppose that the sequence of machines s is fixed for each job j . One machine can do only one job at a time. Several machines cannot do the same job at the same moment. The decision $d_i(j) \in D_i$ means the start of a job $j \in J_i$ at stage i . We define the set of feasible decisions D_i as the set J_i of jobs available at the stage i conforming to the flow-shop rules.

The objective function is the make-span v . Denote by $T_j(d)$ the time when we complete job j (including the gaps between operations) using the decision sequence d . Then the make-span for d is

$$v(d) = \max_{j \in J} T_j(d). \quad (12.8)$$

12.3.2 Algorithm

Permutation Schedule

We can see that the number of feasible decisions for the flow-shop can be very large. The number can be reduced by considering only smaller subset of schedules, the so-called permutation schedules.

The permutation schedule is a schedule with the same job order on all machines. Such a schedule can be defined by fixing job indices $1, 2, \dots, n$. We assume the first operation to be on the first machine, the second on the second, and so on. The schedule is transformed by a single permutation of job indices. It is generally assumed that permutation schedules approach the optimal decision sufficiently closely and are easier to implement (see [6]).

Denote

$$\tau_j = \sum_{s=1}^{|S|} \tau_{j,s},$$

where $|S|$ stands for the number of machines. We define τ_j as the length of the job j .

Heuristics

Define the Longer-Job heuristics satisfying expression (11.2)

$$h_i(j) = \frac{\tau_j - A_i}{A^i - A_i} + a. \quad (12.9)$$

Here

$$A_i = \min_{j \in J_i} \tau_j, \quad A^i = \max_{j \in J_i} \tau_j, \quad a > 0. \quad (12.10)$$

We also consider the well known Gupta priorities (see [6]).

$$t_j = \frac{e_j}{\min_{1 \leq s \leq |S|-1} (\tau_{j,s} + \tau_{j,s+1})},$$

where

$$e_j = \begin{cases} +1, & \text{if } \tau_{j,1} < \tau_{j,|S|} \\ -1, & \text{otherwise.} \end{cases}$$

We define the Gupta heuristics by an expression similar to (12.9)

$$h_i(j) = \frac{t_j - A_i}{A^i - A_i} + a. \quad (12.11)$$

Here

$$A_i = \min_{j \in J_i} t_j, \quad A^i = \max_{j \in J_i} t_j, \quad a > 0.$$

The Longer-Job heuristic (12.9) and the Gupta heuristic (12.11) define job preferences by different priority rules. The priority rule (12.9) prefer a longer job.

The Gupta priority rule (12.11) is the "multi-machine" extension of well known Johnson's rule for the exact solution of the two-machine problem. If there are more than two machines, then the Gupta priority rule (12.11) may be considered as a greedy heuristic, an alternative to the LJ heuristic (12.9).

In both cases a randomized decision function r_i is defined by expression (11.6). We optimize it by solving stochastic optimization problem (11.30) (11.31) where function $f_K(x)$ defines the minimal make-span (see (12.8)) found as a result of K repetitions.

12.3.3 Results

Table 12.7 illustrates the results of the Bayesian method after 100 iterations using the Longer-Job heuristic (12.9) and different randomization procedures. Assume that $J = S = O = 10$, where J , S , O are the number of jobs, machines, and operations, respectively. Lengths and sequences of operations are generated as random numbers uniformly distributed from 0 to 99. The expectations and standard deviations are estimated by repeating optimization of a randomly generated problem 40 times. In Table 12.7 the symbol f_B denotes a mean,

Table 12.7 The results of Bayesian methods using Longer-Job heuristics (12.9)

$R = 100, K = 1, J = 10, S = 10, \text{ and } O = 10$					
Randomization	f_B	d_B	x_0	x_1	x_2
Delta	6.183	0.133	0.283	0.451	0.266
Taylor 3	6.173	0.083	0.304	0.276	0.420
Legendre 3	6.140	0.214	0.321	0.335	0.345
CPLEX	12.234	0.00	—	—	—

and d_B denotes a standard deviation of make-span. "Delta" denotes randomization (11.19), "Taylor 3" denotes randomization (11.6) with the number of terms $N = 3$, "Legendre 3" denotes randomization (11.13) with $N = 3$, and "CPLEX" denotes the results of the well known general discrete optimization software after 2000 iterations (one CPLEX iteration is comparable to a Bayesian observation). The bad results of CPLEX show that the standard MILP technique is not efficient in solving this specific problem of discrete optimization.

It is not yet clear how much one improve the results using specifically tailored B&B.

Table 12.8 shows the results of the Bayesian method after 100 iterations using the Gupta heuristic (12.11), the Longer-Job heuristic (12.9), the Longer-Remaining-Time (LRT) heuristic (12.12), and different randomization procedures. We also included the results of deterministic techniques, for comparison. The number of jobs $J = 7$, the number of machines $S = 25$, and the number of operations $O = 25$.

We define lengths and sequences of operations generating random numbers uniformly distributed from 0 to 99, and estimate expectations and standard deviations repeating optimization of 10 randomly generated problems 40 times. No comparisons with CPLEX are made in Table 12.8, since the example is too large. In Table 12.8 the notation "Taylor 4" means randomization (11.6)

Table 12.8 The results of Bayesian methods using Gupta (12.11), Longer-Job (12.9), and Longer-Remaining-Time (12.12) heuristics

$R = 100, K = 1, J = 7, S = 25, \text{ and } O = 25$							
Randomization	f_B	d_B	T	x_0	x_1	x_2	x_3
Gupta heuristics (see expression (12.11))							
Delta	1861.38	16.89	—	0.383	0.349	0.267	—
Taylor 3	1857.38	11.90	—	0.008	0.365	0.625	—
Taylor 4	1857.02	16.27	—	0.252	0.271	0.224	0.253
Legendre	1870.21	15.16	—	0.386	0.377	0.238	—
Deterministic	2092	—	—	—	—	—	—
Longer-Job heuristics (see expression (12.9))							
Delta	1876.81	20.68	—	0.420	0.341	0.239	—
Taylor 3	1868.67	18.13	149	0.387	0.308	0.305	—
Taylor 4	1874.82	16.52	199	0.277	0.263	0.231	0.229
Legendre	1856.85	17.48	149	0.311	0.326	0.363	—
Deterministic	2261	—	—	—	—	—	—
Longer-Remaining-Time heuristics (see expression (12.12))							
Delta	2113.50	47.59	—	0.041	0.023	0.934	—

with the number of terms $N = 4$, "Deterministic" denotes a choice of the best heuristic, without any randomization, and T denotes the CPU time.

The LRT heuristics $h_i(j)$ depends on the remaining time τ_j of the job j and is defined by expression

$$h_i(j) = \frac{\tau_i(j) - A_i}{A^i - A_i} + a. \quad (12.12)$$

Here

$$A_i = \min_{j \in J_i} \tau_i(j), \quad A^i = \max_{j \in J_i} \tau_i(j), \quad a > 0$$

and

$$\tau_i(j) = \sum_{s \in S(i,j)} \tau_{j,s},$$

where $S(i, j)$ is the set of machines to be used after the stage i to complete the job j .

Assuming that a machine can be used only once, $S(i, j)$ is the set of machines not belonging to the set of machines used by the previous decisions d^i .

Thus the LRT heuristics prefer jobs with the longest remaining time at each stage i .

We see that the Bayesian adaptation of heuristics significantly improved the results in all the cases. The Gupta heuristic (12.11) provides better results compared to the LJ heuristic (12.9). Unexpectedly the number of terms and orthogonality of polynomials shows no significant effect. We did expect better results from the LRT heuristics. Disregarding those two exceptions, the results of Table 12.7 and Table 12.8 in general correspond to the theoretical predictions.

Usually we compare the Bayesian and other methods neglecting the "learning" potential of Bayesian methods. For example, Figure 12.1 (top) estimates the learning effect (see Section 3.4) by showing the density of optimal values of parameters x_1 and x_2 . Those values correspond to 100 different randomly generated flow-shop problems. We see that the maximal density of optimal values of x_1 and x_2 exceeds the average density several times. According to the results of Section 3.4 it means that the optimal parameters of Bayesian methods are rather robust and thus could be applied to other related problems.

For a different approach to the flow-shop problem, see [145].

12.3.4 Stochastic Flow-Shop Problem

Consider the case when some machine can fail. Thus we obtain not one but $L + 1$ scheduling problems $\theta_l, l = 0, \dots, L$ where L is the number of machines.

Denote by $\theta_l, l = 1, \dots, L$ the case when all machines are working with the exception of machine j . Denote by θ_0 the case when all machines are functioning. We define each problem θ_l by a different operation time matrix $\tau_{j,s}(l), l = 0, 1, \dots, L, j \in J, s \in S(l)$. Here $S(l) = S_l$ if $l = 1, \dots, L$ and $S(l) = S$, if $l = 0$

The parameter q_l defines the probability of the case $\theta_l, l = 0, 1, \dots, L$, occurring. The parameter q_0 denotes the probability that all machines work. Neglecting the events when more than one machine is out of order

$$q_0 = 1 - \sum_{l=1}^L q_l. \quad (12.13)$$

The make-span is hardly a compatible objective function for the stochastic problem. The total throughput time (the time in the shop for all the jobs) is a better objective function for the stochastic scheduling problem.

Denote by $f_K(x, l)$ the best value of the throughput time obtained after K repetitions of the randomized decision procedure $r(x)$ for fixed x and θ_l . Define the average value of $f_K(x, l)$ by expression (11.33)

$$f_K(x) = \frac{1}{L+1} \sum_{l=0}^L \frac{1}{K(l)} \sum_{k=1}^{K(l)} f_K(x, l) \quad (12.14)$$

Define the number of averaging repetitions $K(l)$ by expression (11.34)

$$K(l) = q_l K_0 \quad (12.15)$$

Here K_0 is the number of averaging repetitions corresponding to the minimal probability $q_l, l = 0, 1, \dots, L$. The number K_0 should not be too small.

We may use the same LJ heuristics (12.9) as in the deterministic scheduling problem. However, the length of the job j is a less obvious heuristic, if the objective function is the throughput time. We need further study to define appropriate greedy heuristics for the stochastic flow-shop problem.

The randomized decision function r_i is defined by expression (11.6). It is optimized by solving stochastic optimization problem (11.30) (11.31) where function $f_K(x)$ is the the best average value of throughput time obtained after K repetitions (see (12.14)).

We can consider the on-line scheduling problem by defining the corresponding sequence of stochastic scheduling problems.

12.3.5 "Learning" Bayesian Heuristics

Introduction

In this section the "learning" mode using BHA is considered. The "learning" mode means that the randomization parameters are optimized for some "learning" set of problems. These parameters are used later on for a family of related problems. The "non-learning" form corresponds to the case in which the randomization parameters are optimized for each problem separately.

We define the learning efficiency as a non-uniformity of the optimal parameters while solving a set of randomly generated problems. We optimize the randomization parameters by multiple application of randomized heuristics. One may optimize these parameters for each problem separately. Alternatively one may optimize the randomization parameters only for some "learning" set of problems. The "learned" parameters may be used later on without or with an additional optimization.

The question is whether such learning will help. We try to obtain the answer by Monte Carlo simulation of a set of flow-shop problems using randomized heuristics.

Learning is regarded as ineffective, if the density of optimal values of parameters is uniform. Therefore we define the "non-uniformity" of optimal parameters as a measure of learning efficiency, see Section 3.4. We show that for flow-shop problems non-uniformities and consequently learning is significant.

Learning Mode

We generate 100 flow-shop problems with 10 tasks and 10 tools each [10]. We define lengths and sequences of operations by random numbers uniformly dis-

tributed from 0 to 99. 100 different problems are generated and randomization parameters of each problem are optimized separately by Bayesian techniques. The first three figures show the density of the optimal parameters.

Figure 12.1 (top) shows the probability density function of the first parameter pair (x_1, x_2) . Figure 12.1 (bottom) shows the density function of the second pair (x_1, x_3) . Figure 12.2 shows the probability density function of the third pair (x_2, x_3) . The last three figures show the objective as a function of the first pair of parameters (x_1, x_2) for three different samples of the flow-shop problem. The density functions and objectives are smoothed to improve the visualization. For the densities (see the first three figures) the following smoothed function is used:

$$T(x) = 1/N \sum_{i=1, N} e^{-C_k |x_i - x|}. \quad (12.16)$$

Here $T(x)$ denotes a smoothed density of the optimal parameters, N is the number of points, and $C_1 = C_2 = C_3 = 15$, $C_4 = 22$, $C_5 = 18$, $C_6 = 22$ are smoothing parameters.

The objectives were smoothed (see the last three figures) using a smoothing function (12.17)

$$F(x) = \frac{1}{\sum_{i=1, N} f(x_i) e^{-C_k |x_i - x|}} \sum_{i=1, N} e^{-C_k |x_i - x|}. \quad (12.17)$$

Here $F(x)$ denotes a smoothed density of the objective function.

Figures 12.1 and 12.2 show that in a flow-shop problem the learning of BHA is rather good because the density of optimal parameters is considerably greater around the point $x = (0.2, 0.5, 0.6)$.

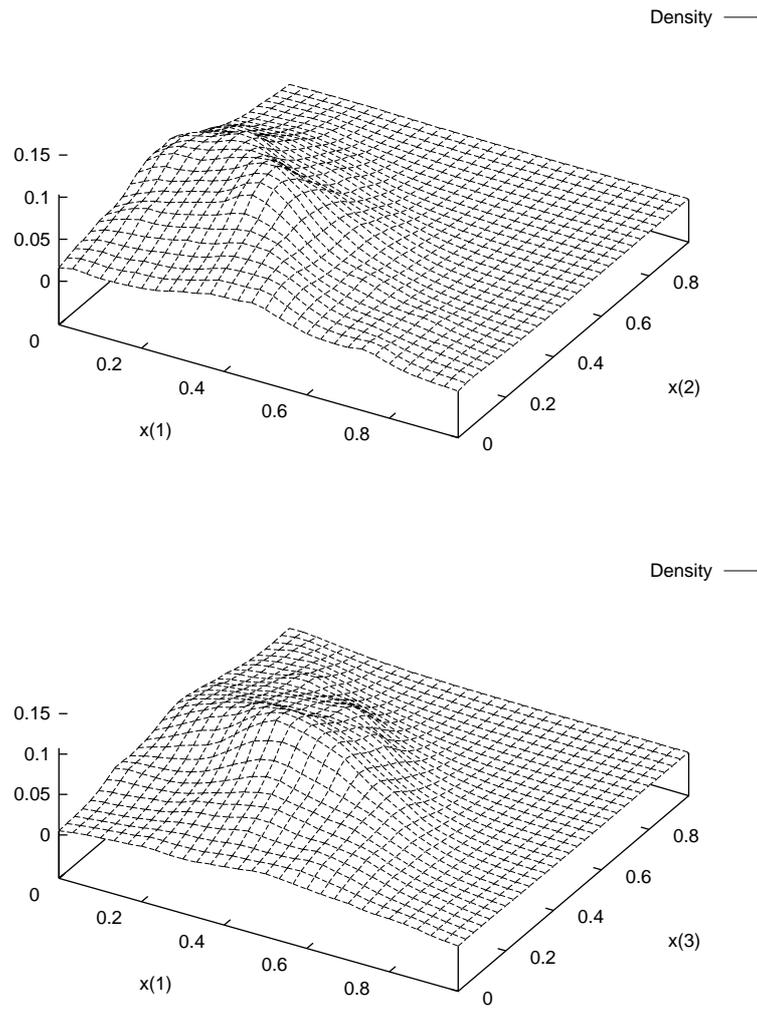


Figure 12.1 Density of optimal parameters x_1, x_2 (top), and that of x_1, x_3 (bottom)

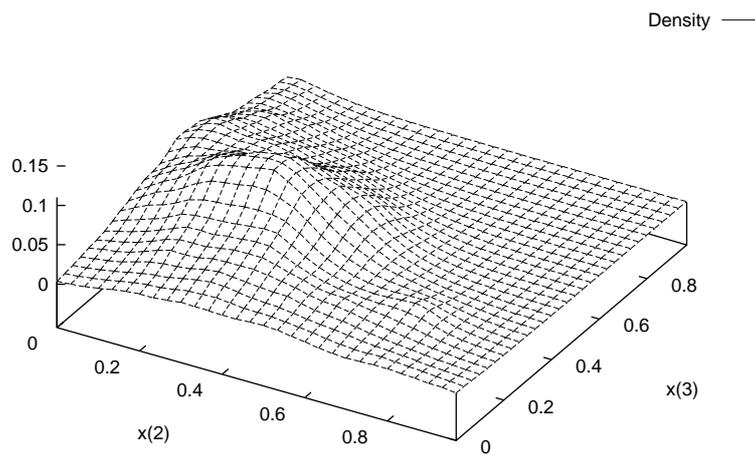


Figure 12.2 Density of optimal parameters x_2, x_3

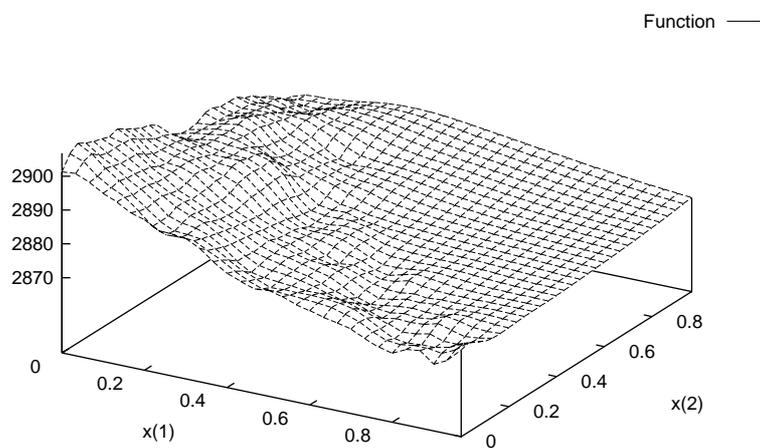


Figure 12.3 The objective of the 1-st sample problem as a function of parameters x_1, x_2

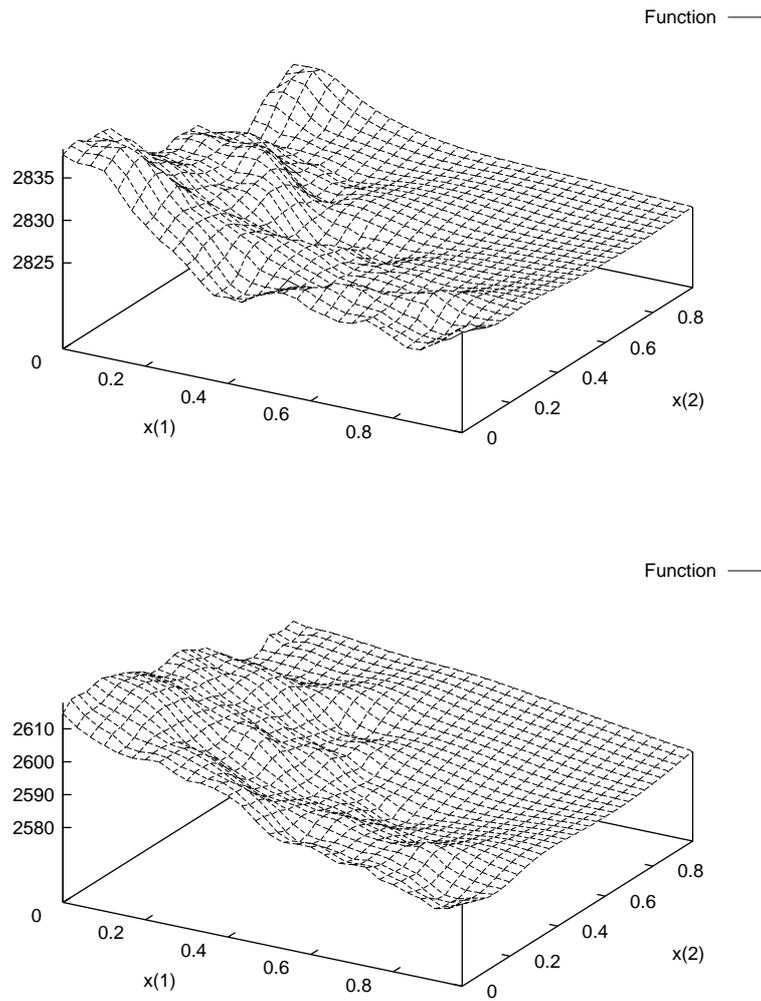


Figure 12.4 The objective of the 2-d sample problem as a function of parameters x_1, x_2 (top), and that of the 3-d sample function

12.4 JOB-SHOP PROBLEM

This discrete optimization problem is stated in a similar way to the flow-shop problem. The difference is that we do not fix the sequence of machines required to execute job j . At each stage we choose not only the next job $j = j_i$ but also the next machine $s = s_i$ to do the chosen job j_i .

Here the greedy heuristic has two components $h_i(m) = (h_i(j), h_{(j_i, s)})$, $m = (j, s)$. The first component $h_i(j)$ depends on the remaining time τ_j of the job j . It may be defined by the expression

$$h_i(j) = \frac{\tau_i(j) - A_i}{A^i - A_i} + a.$$

Here

$$A_i = \min_{j \in J_i} \tau_i(j), \quad A^i = \max_{j \in J_i} \tau_i(j), \quad a > 0,$$

and the remaining time at stage i

$$\tau_i(j) = \sum_{s \in S(i, j)} \tau_{j, s}, \quad (12.18)$$

where $S(i, j)$ is the set of machines to be used after the stage i to complete the job j .

Assuming that a machine can be used only once, then $S(i, j)$ is the set of machines not belonging to the set of machines used by the previous decisions d^i . Thus the first component is a heuristic of LRT type preferring the jobs with the longest remaining time at each stage i .

The second component $h_{(j_i, s)}$ depends on the time $\tau_{j_i, s}$ of operation (j_i, s) . It can be expressed as

$$h_i(j_i, s) = \frac{\tau_{j_i, s} - C_i}{C^i - C_i} + a. \quad (12.19)$$

Here

$$C_i = \min_{s \in S_i} \tau_{j_i, s}, \quad C^i = \max_{s \in S_i} \tau_{j_i, s}. \quad (12.20)$$

We apply the randomized procedure similar to (11.6) twice. The first time the next job $j = j_i$ is defined by the heuristic $h_i(j)$. The second time the next machine $s = s_i$ is defined for the fixed job $s = s_i$ by the heuristic $h_i(j_i, s)$. The randomized decision procedure $r(x)$ is optimized by solving the stochastic optimization problem (11.30) (11.31) where function $f_K(x)$ defines the minimal make-span found after K repetitions at fixed vector x .

For different approaches to solving the job-shop problem, see [107].

12.5 PARAMETER GROUPING

12.5.1 Definition

Parameter grouping is important in cluster analysis and empirical data processing (see [2]).

A good partition means:

- strong interaction inside the groups
- weak interaction outside the groups.

The partition is described by a discrete vector $d = (d(1), \dots, d(n))$ where $d(i) = j$ means that the parameter s_i belongs to the group g_j . We maximize the sum of parameter-group interactions using permutations (see [33]). The permutation m means adding or subtracting a unit to $d(i)$ Only one component is changed at a time (see [101]). The search is terminated after k steps. We optimize the initial "temperature" x of Simulated Annealing (see(11.23)). The one-dimensional Bayesian algorithm (4.10) is used.

12.5.2 Results

The number of randomly generated problems is 20. The number of iterations is 20. The initial partition is:

$$g_1 = \{s_1, \dots, s_5\}, g_2 = \{s_6, \dots, s_{10}\}, g_3 = \{s_{11}, \dots, s_{15}\}, g_4 = \{s_{16}, \dots, s_{20}\}.$$

The optimal value $x^* = 0.08$.

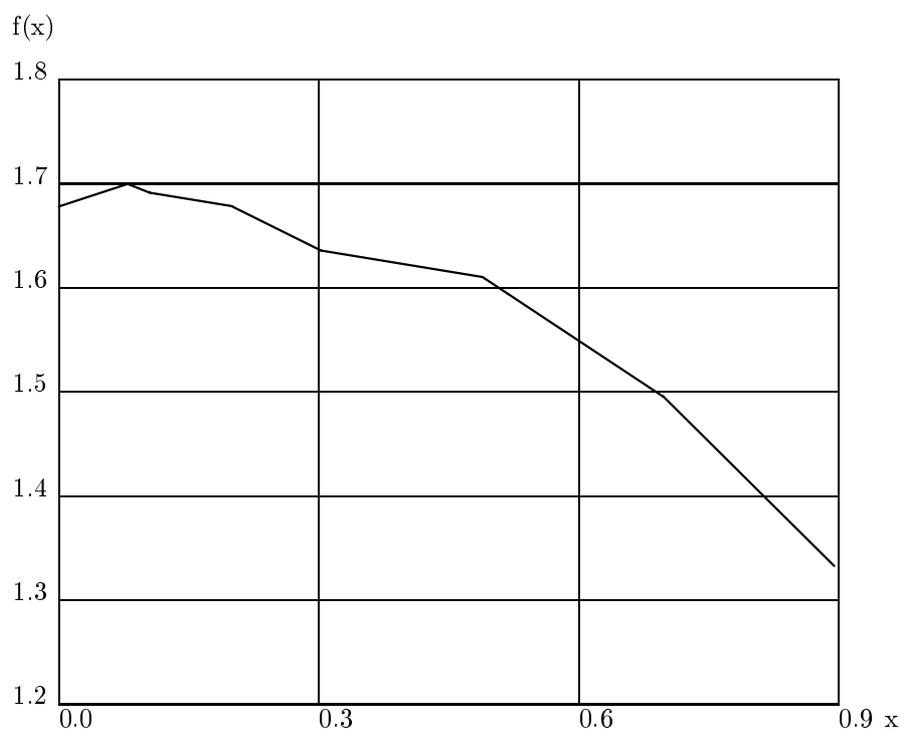


Figure 12.5 The relation of simulated annealing results with the initial temperature x

12.6 CONCLUSIONS

The discrete optimization problem is considered as a multi-stage decision one. We define the probabilities of a randomized decision procedure $r(x)$ as a function of heuristics h and the parameter x . The randomized search is repeated K times. The result is denoted as a stochastic function $f_K(x)$. Thus one can obtain the global minimum of the discrete problem, if the number K is sufficiently large.

We optimize parameters x to make the randomized decision procedure $r(x)$ more efficient. That is done by global stochastic optimization techniques such as the Bayesian methods of global optimization. Thus we obtain the global minimum, if the number $K_B = KR$ is sufficiently large, where R is the number of iterations of the optimization method. If this number is not large, an approximation to the global minimum is obtained.

This is a new approach which helps to incorporate, in a natural way, expert knowledge about a specific problem into the mathematical framework of optimization. It is well known that expert knowledge can be very important, if we wish to obtain an approximate solution fast (see [156, 157, 122, 83, 5]).

Thus the main results of BHA are:

- the expert knowledge is included in the heuristics h ;
- the convergence is provided by randomization $r = r(m, h, x)$;
- the efficiency of search is increased by optimizing $r(m, h, x)$ via Bayesian techniques of continuous optimization;
- the optimal x may be applied to a family of related problems.

13

APPLICATION OF BHA TO MIXED INTEGER NONLINEAR PROGRAMMING (MINLP)

13.1 INTRODUCTION

An important part of nonlinear optimization is Mixed Integer Nonlinear Programming (MINLP). MINLP involves both integer and real variables. Traditionally algorithms of B&B type are used for these problems. The B&B algorithms generally needs exponential time to ensure the exact solution. Thus application of BHA is desirable if approximate solution is acceptable and exponential processing time is not.

The well known subset of MINLP family is Mixed Integer Linear Programming (MILP). In this chapter in solving MILP problems we will apply BHA using heuristics involved in Genetic Algorithms (GA). The reason is that GA is developing fast and provides an interesting family of randomized heuristics.

In solving Mixed Integer Bilinear Programming (MIBLP) problems BHA is applied using penalty function type of heuristics (PFH). Both GA and PFH are rather general heuristics. However, domain oriented heuristics may be more efficient in solving specific MINLP problems. An example is MRP heuristics (see Chapter 14).

13.2 APPLYING OF GENETIC ALGORITHMS (GA) TO MILP PROBLEM

13.2.1 General Idea

The general idea is to reduce the original MILP problem by fixing some integer variables. We define the reduced MILP as the "heuristic MILP" and the fixed variables are referred to as the "heuristic variables". We stop optimizing the heuristic MILP as soon as the first feasible solution is obtained. This solution is defined as "heuristic incumbent" and the corresponding objective is regarded as heuristics.

If we prefer the standard Linear Programming (LP) software (instead of MILP), then we may create the corresponding "heuristic LP" problem by fixing "heuristic variables" and "relaxing" the remaining ones. If the feasible solution may be obtained and the corresponding heuristics defined by rounding-off the relaxed variables then the "heuristic LP" could be more efficient as compared to the "heuristic MILP". A description of the m -th step of the "heuristic LP" algorithm is as follows:

1. fix the heuristic variables $s(m) = (s_i(m), i = 1, \dots, I(m))$, $m = 1, \dots, M$;
2. relax the remaining integer variables $z = (z_j, j = I(m) + 1, \dots, J)$;
3. denote the LP solution under these conditions as $z(m) = (z_j(m))$ and $y(m) = (y_l(m), l = 1, \dots, L)$, where y is a vector of continuous variables;
4. obtain a feasible solution by rounding $z(m)$ components;
5. define heuristics as an objective function $L(m)$ at the feasible solution $h(m) = -L(m)$ (assuming that greater $h(m)$ is better);
6. randomize heuristics $h(m)$ by the procedure satisfying convergence conditions (3.2.1) (assuming that $a \leq h(m) \leq b$, $a > 0$);
7. select vectors $s(m)$ using this randomized procedure;
8. update the best feasible solution;
9. optimize the parameters of randomization and heuristic generation.

Both heuristic MILP and heuristic LP procedures are based on the assumption that one may obtain a feasible MILP solution by an algorithm of polynomial complexity. Otherwise, one has to design an algorithm to obtain approximately a feasible solution, for example, by using a penalty function. Using penalty functions, we have to take care that in accepting the approximately feasible solutions we do not violate convergence conditions (3.2.1).

13.2.2 Randomization by GA

A procedure that generates and selects a sequence of decision vectors $s(m)$ is defined as randomization. Genetic Algorithms (GA) present a general and visual way of randomization in this sense.

GA is a general methodology for searching a solution space in the manner similar to the natural selection procedure in biological evolution (see [62]). Each candidate solution is represented by a string of symbols. The set of solutions at state m is referred to as the population of the m th generation. The population evolves for a prescribed number M of generations.

The basic structure processed by GA is the string. Strings are composed of a sequence of characters and may be represented as follows

$$s(m) = (s_i(m), i = 1, \dots, I), \quad m = 1, \dots, M \quad (13.1)$$

where $s_i(m)$ is a character of length λ .

A simple GA consists of one reproductive plan, defined as the fitness proportional reproduction and two genetic operators, defined as cross-over and mutation. The probability of selection is defined (see [53]) as

$$r(m) = h(m) / \sum_m h(m) \quad (13.2)$$

During the cross-over operation we split two selected strings at some random position i_s . Then two new strings are created, by exchanging all the characters up to the split point i_s .

During the mutation operation we alter some string characters at random. A mutation is said to be of n -th order if we change n elements during one mutation operation. A mutation is feasible if it satisfies all the constraints.

The traditional definition of simple GA does not determine the rates of cross-over and mutation operations and considers only linear randomization (fitness proportional reproduction).

In the BHA framework these rates are regarded as unknown parameters of the heuristic generation procedure. We consider a mixture of different randomization procedures (see, for example, a "polynomial" set (11.6) that also includes linear randomization (13.2)). The optimal rates and the optimal polynomial randomization parameters are obtained by BHA.

One may obtain different heuristic algorithms merely by a change of the GA parameters. For example, GA may be reduced to simulated annealing if we

- restrict the population size to merely two members: the old m and the new one $m + 1$;
- fix zero cross-over rate;
- replace the fitness proportional reproduction (13.2) by fitness exponential reproduction (13.3)

$$r(m) = \begin{cases} e^{\frac{h(m+1)-h(m)}{x/\ln(1+m)}}, & \text{if } h(m+1) < h(m) \\ 1, & \text{otherwise,} \end{cases} \quad (13.3)$$

where parameter x denotes an initial temperature.

13.2.3 Advantages and Disadvantages of GA while using BHA

To satisfy convergence conditions (see Theorem 3.2.1) mutations are needed. A simple explanation is that mutations help us to escape from "local" minima. The need for cross-over is not so clear.

Traditional GA assume that some segments of strings define specific "traits". In such a case we may "enforce" good traits by uniting "good" segments. One may also expose bad traits by uniting "bad" segments. The reproduction plans favor the good traits and tend to reject the bad ones. Thus, cross-over will "improve" the population, if the traditional assumption is true.

If not, then the cross-over may be regarded merely as a sort of mutation that may help to jump the area dividing separate "local" minima. The same jump

may be accomplished by high order mutation, too. However, the cross-over operation may work better in some cases.

Traditionally (see [4]) mutation is regarded as a more "radical" operation as compared with cross-over. This is true if we change many elements of "genetic" sequence during one mutation, meaning that the mutation order n is close to the string length I . In this case it may be useful to lower the mutation rate during optimization (see [4]).

The results of some real life network optimization (see [98]) and parameter grouping (see [33]) problems show that better results are reached by low order mutations when merely a few elements of the decision vector $r(k)$ are changed. In such a case, the mutation may be considered as a less radical operation because less elements of the string $r(k)$ are changed as compared with a cross-over operation.

13.2.4 Cross-Over v.s. High-Order Mutation

It is interesting to know how best to escape the "local" minimum: by cross-over or by high-order mutation.

In the latter case the "genetic" analogy may be replaced by a similarity with a "multi-start" algorithm (see [63]) when the local search is applied multiple times from randomly chosen starting points. The low order mutations play a role similar to that of the local search, the high order mutations generate a sort of "starting points".

Apparently the answer depends on the degree of "structure". In the "well-structured" problems one may expect good results using cross-over. It is not clear how cross-over can help in "ill-structured" problems. By "well-structured" we understand such problems, where the objective and/or constraints depend not as strongly on the string characters as on some segments that may be regarded as "trait carriers". That is typical of living organisms but not so typical of discrete optimization problems. If the string contains no such "trait carriers", then an "ill-structured" problem is obtained.

13.2.5 Cross-over Rate Optimization

In the above discussion we have offered no simple test for determining which problem is "well-structured" and which is not. We may obtain a partial answer and improve the efficiency of search by using the cross-over rate as an optimization variable in the BHA framework. Using BHA the cross-over rate may be regarded as a time function (see [4]) and the parameters of this function optimized. This concept may be extended to a "mixture" of time-functions and the mixture parameters optimized also. We denote the above approach as BHA-GA. This approach will be illustrated by application to the Batch Scheduling Problem (see Chapter 16).

13.3 APPLYING PENALTY FUNCTION HEURISTICS (PFH) TO MIXED INTEGER BILINEAR PROGRAMMING (MIBLP)

13.3.1 General Idea

A natural and important extension of MILP is Mixed Integer Bilinear Programming (MIBLP). By the term "Bilinear" we mean a minimization of the sum of products of two different variables. By the term "Mixed Integer" we mean that some of the variables are "Boolean", merely zero or one. Denote $x = (x_i, i = 1, \dots, n_x)$, $w = (w_i, i = 1, \dots, n_w)$ to be real valued variables and $y = (y_i, i = 1, \dots, n_y)$, $z = (z_i, i = 1, \dots, n_z)$ to be Boolean variables.

Suppose that the real variables $w = (w_i, i = 1, \dots, n_w)$ multiply only by real variables. The Boolean variables z multiply only by Boolean variables. This partition of variables is merely for convenience in problem formulation. Suppose, for simplicity of description, that $n_x = n_w = n_y = n_z = n$. Then we minimize

$$\begin{aligned} \min_{x,w,y,z} & \left(\sum_{i=1,n} (a_{0i}x_i + b_{0i}y_i + c_{0i}z_i + d_{0i}w_i) + \right. \\ & \left. \sum_{i,j=1,n} (a_{0ij}x_iy_j + b_{0ij}y_iz_j + d_{0ij}x_iw_j) \right), \end{aligned} \quad (13.4)$$

satisfying constraints

$$\begin{aligned} & \left(\sum_{i=1,n} (a_{ki}x_i + b_{ki}y_i + c_{ki}z_i + d_{ki}w_i) + \right. \\ & \left. \sum_{i,j=1,n} (a_{kij}x_iy_j + b_{kij}y_iz_j + d_{kij}x_iw_j) \right) \geq 0, \quad k = 1, \dots, m \quad (13.5) \\ & x_i \geq 0, w_i \geq 0, y_i \in \{0, 1\}, z_i \in \{0, 1\}. \end{aligned}$$

Assume that $a_{ki} \neq 0, b_{ki} \neq 0, d_{ki} \neq 0, c_{ki} \neq 0, k = 0, \dots, m, i = 1, \dots, n$. Also suppose that most of the triple-index parameters such as $a_{kij}, b_{kij}, c_{kij}, d_{kij}$ are zero. Otherwise, too many additional inequalities could be needed later, when reducing MIBLP to the standard Mixed Integer Linear Programming (MILP).

13.3.2 Solution of MIBLP as a sequence of Mixed Integer Partly Bilinear Programming problems (MIPBLP)

By the term "Partly Bilinear" we mean a problem in which there are no products of real variables. That helps us when reducing MIPBLP to the standard MILP. Formally one can define MIPBLP merely by deleting the variables w in (13.4),(13.5). We minimize

$$\begin{aligned} \min_{x,y,z} & \left(\sum_{i=1,n} (a_{0i}x_i + b_{0i}y_i + c_{0i}z_i) + \right. \\ & \left. \sum_{i,j=1,n} (a_{0ij}x_iy_j + b_{0ij}y_iz_j) \right), \quad (13.6) \end{aligned}$$

satisfying constraints

$$\begin{aligned} & \left(\sum_{i=1,n} (a_{ki}x_i + b_{ki}y_i + c_{ki}z_i) + \right. \\ & \left. \sum_{i,j=1,n} (a_{kij}x_iy_j + b_{kij}y_iz_j) \right) \geq 0, \quad k = 1, \dots, m \quad (13.7) \\ & x_i \geq 0, y_i \in \{0, 1\}, z_i \in \{0, 1\}. \end{aligned}$$

MIBLP is solved as a sequence of corresponding MIPBLP problems by the following iterative algorithm:

1. fix some initial value of $w = w^0$. in expressions (13.4), (13.5);

2. determine the optimal value $x = x^0$, given $w = w^0$;
3. determine the optimal value $w = w^1$, given $x = x^0$;
4. stop, if there is no improvement, otherwise go to step 1.

A disadvantage of this technique is that for some initial $w = w^0$ there might be no feasible solutions. We may succeed if we carefully select w^0 . Otherwise we relax the constraints and minimize the auxiliary objective. The auxiliary objective includes the original one, plus the distance to the original feasible set. We shall describe the auxiliary objective later on, when considering Combinatorial Linear Programming (CLP).

An alternative technique is the Outer Approximation algorithm (see [128]). However the Outer Approximation is rather a general technique. It can be expected to be less efficient as compared to the algorithm, designed specially for MIBLP.

13.3.3 Reduction of MIPBLP to MILP

There are well known algorithms for MILP problems. We reduce here the MBPBLP problem to the MILP problem. We minimize

$$\min_{x,y,z,u,v} \left(\sum_{i=1,n} (a_{0i}x_i + b_{0i}y_i + c_{0i}z_i) + \sum_{i,j=1,n} (a_{0ij}v_{ij} + b_{0ij}u_{ij}) \right), \quad (13.8)$$

satisfying constraints

$$\left(\sum_{i=1,n} (a_{ki}x_i + b_{ki}y_i + c_{ki}z_i) + \sum_{i,j=1,n} (a_{kij}v_{ij} + b_{kij}u_{ij}) \right) \geq 0, \quad k = 1, m \quad (13.9)$$

$$v_{ij} \leq Ky_j, \quad u_{ij} \leq y_i, \quad u_{ij} \leq z_j$$

$$u_{ij} \geq y_i + z_j - 1$$

$$x_i \geq 0, v_{ij} \geq 0, y_i \in \{0, 1\}, z_i \in \{0, 1\}, u_{ij} \in \{0, 1\},$$

where K is a large number.

Problem (13.8), (13.9) can be solved using a MILP algorithm. However, we have to stop before reaching the global minimum being short of time in many real problems. In such cases the "incumbent" may be regarded as an approximate solution of MILP. Such a technique is referred to as the "truncated B&B".

The truncated B&B looks like a very convenient deterministic technique. We proceed until the time limit is reached. Then we stop and obtain an approximate solution. However B&B is designed to obtain the exact solution. It is not clear how good a solution "incumbent" really is. The convergence to the exact solution does not necessarily mean that a good solution will be obtained, if we stop before reaching the exact solution, see, for example, Table 12.7.

A convenient way to obtain a good approximation is by designing randomized heuristics. The heuristics helps to involve the expert intuition. Randomization gives the flexibility of design. Optimization of the decision parameters adapts the algorithm to the given family of problems.

13.3.4 Reduction of MIPBLP to Combinatorial Linear Programming (CLP)

For problems in which it is too difficult to keep inside the feasible regions penalty function construction can prove to be useful. The penalty function may be regarded as a sort of heuristics which is defined as follows: fix the values of discrete variables y, z and minimize the original linear objective (13.6) plus the sum of violations of constraints (13.7) multiplied by the factor $r > 0$. The result is denoted by $L(y, z)$. From this definition and expressions (13.6), (13.7) it follows that

$$L(y, z) = \min_x \left(\sum_{i=1, n} (a_{0i}x_i + b_{0i}y_i + c_{0i}z_i) + \sum_{i, j=1, n} (a_{0ij}x_i y_j + b_{0ij}y_i z_j) + r \sum_{k=1, m} s_k \right), \tag{13.10}$$

where

$$s_k \geq - \left(\sum_{i=1, n} (a_{ki}x_i + b_{ki}y_i + c_{ki}z_i) + \sum_{i, j=1, n} (a_{kij}x_i y_j + b_{kij}y_i z_j) \right), \quad k = 1, \dots, m \tag{13.11}$$

$$x_i \geq 0, s_k \geq 0, y_i \in \{0, 1\}, z_i \in \{0, 1\}.$$

The advantage of CLP is that all the Boolean decisions y, z are "feasible". Here the "feasibility" of y, z means that some auxiliary objective $L(y, z)$ is defined for any fixed y, z . We see from (13.10) and (13.6) that the auxiliary objective

(13.10) is the minimum of the original objective (13.6) plus the "distance" from the original feasible set (13.7). The distance from set (13.7) is defined as the sum of constraint violations s_k . Now let us apply some randomized heuristics to minimize $L(y, z)$ as a function of Boolean variables y, z . We consider an algorithm that solves CLP in five steps:

1. fix a vector (y^0, z^0) ;
2. generate a "perturbation" vector (y^l, z^l) ;
3. define heuristics h_l for the perturbation (y^l, z^l) . A simple and natural way is to assume the heuristics h_l to be a function (13.12) of the auxiliary objective $L_l = L(y^l, z^l)$;
4. go over to perturbation l with probability r_l defined as some function of heuristics h_l ;
5. repeat the randomized technique many times optimizing the parameters of the randomization function by Bayesian methods (see [94]).

Define the heuristics h_l as

$$h_l = L_l - A + a, \quad (13.12)$$

where

$$A = \min_{j \in J} L_j, \quad a > 0.$$

Here J is a set of perturbations.

The simplest example of this algorithm is to perturb the Boolean variables $(y, z) = (y_i, z_i, i = 1, \dots, n)$ one-by-one, then to use the simulated annealing, to optimize the unknown parameters (T_0, r) by the Bayesian techniques. Here T_0 is the initial temperature of annealing. We can also use more complicated perturbation and randomization algorithms, described by [94].

The heuristics h_l are designed for a family of MIPBLP problems. For more specific problems one can design simpler heuristics tailored to fit only those problems. For example, the longest remaining time for a given job is apparently the simplest heuristics in some scheduling problems.

It is important to investigate which approximation is better: solving MIPBLP by the truncated B&B or solving CLP by Bayesian optimization of randomized heuristics. To encounter similar problems while considering the scheduling of batch operations (see Chapters 14, 15, and 16).

PART V

BATCH PROCESS SCHEDULING

14

BATCH/SEMI-CONTINUOUS PROCESS SCHEDULING USING MRP HEURISTICS

14.1 INTRODUCTION

The problem of decision timing in the context of batch scheduling is addressed in this part of the book. Representation of time in any scheduling model affects the number of integer variables and the modality of the objective function ¹.

The traditional procedure in batch process scheduling is to divide the scheduling interval into equal-size intervals short enough to achieve the required accuracy. That is the Uniform Discrete-Time Model(UDM). This construction generates a formulation with a potentially very large number of Boolean variables².

In the following the time events arising in the schedule will be optimized directly. A Non-Uniform Discrete-Time Model (NUDM)³ excludes Boolean variables over periods during which no changes occur in the system state.

Using several test examples computational comparisons are made against the UDM formulation. The results suggest that BHA combined with the NUDM looks promising for the solution of batch scheduling problems.

¹Meaning, for example, uni-modal or multi-modal functions.

²The term "Boolean" denotes zero-one variables. The term "binary" is reserved for two-valued variables, not necessarily zero-one.

³In this work time events are represented by real numbers therefore the term "Discrete" may be omitted.

14.1.1 Different Ways to Apply BHA

We apply BHA to the Batch Scheduling Problem in three different ways:

- using a specific heuristics, such as the well known Material Requirements Planning (MRP) (see this chapter) while formulating the problem as a Mixed Integer Bilinear Program (MIBLP) (see Chapter 13);
- optimizing "initial temperature" of Simulated Annealing while applying the MIBLP penalty function as a heuristic (see Chapter 15).
- applying a Genetic Algorithm (GA) and optimizing its parameters while considering GA as randomized heuristics (see Chapter 16). Using GA it is convenient to reduce MIBLP to Mixed Integer Linear Programming (MILP). This way involves a very large system of equations and inequalities, but we may easily extend the results to any other MILP problem.

The computational results of both the MRP and the Simulated Annealing cases are described in Chapter 15.

14.1.2 Why Consider Batch Scheduling?

A wide range of products in the chemical processing industry are produced using the batch mode of production. This mode of production has long been the accepted procedure for the manufacture of many types of chemicals (specialty chemicals, pharmaceuticals, polymers, biochemical, and foods), particularly those which are produced in small quantities and for which the production processes or the demand pattern are likely to change.

The most important feature of batch processes is their flexibility in processing multiple products by accommodating diverse operating conditions associated with each product. Therefore, in spite of the traditional drive towards continuous production, the batch mode remains the only alternative for a number of sectors of the processing industry. As a result, there has been an increasing interest in the development of procedures for scheduling batch process operations.

14.2 DIFFERENT MODELS

14.2.1 State-Task Networks

A *state-task* network concept is employed to represent various models of the batch processing system [80]. The basic idea of this concept is to represent the batch processing system as a network of state nodes (which represent the state of material: feed, intermediate and final products) and task nodes (which represent the processing operations). The processing operations transform material from one or more input states to one or more output states. State and task nodes are denoted by circles and rectangles, respectively. Batch sizes and task-equipment allocations are flexible. Temporary unavailability of equipment due to maintenance or breakdown, sequence dependent clean-out times and frequency of use-dependent cleaning of equipment are accommodated directly. This form of NUDM proved to be superior over UDM when processing times differ by, say, an order of magnitude.

14.2.2 Uniform Discrete-Time Models

A well-known approach to batch scheduling problems is a uniform discrete-time model [80]. In this case, to represent a process accurately one needs a large number of Boolean variables.

14.2.3 Non-Uniform Discrete-Time Models

The basic NUDM idea (see [105, 106, 97]) is to divide the scheduling horizon into intervals of unknown length. Then we force each event, such as the start or the end of the task, to occur on the boundaries of these intervals. Some Boolean variables associated with each interval are start indicators. The start indicator is equal to one only when the task starts. Thus, the number of variables depends only on the number of tasks. The number of tasks is smaller than the number of UDM uniform intervals, since the processing times of different tasks are not the same.

In [105, 106, 97], regarding NUDM and assuming the fixed recipes and the constant processing times the problem is posed as a Mixed Integer Bilinear Program (MIBLP). The reason is that the recipes with constant processing times and the *state-task* recipe representation lead to the formulation in which the

nonlinearity is expressed as a product of continuous variables. Batch sizes and task-equipment allocations are not fixed, a temporary unavailability of equipment due to maintenance or breakdown, sequence dependent clean-out times and frequency of use-dependent cleaning of equipment are accommodated directly.

In [105] the non-linear model is reduced to a bilinear one which is made linear using specific linearization techniques. The Outer Approximation algorithm (OA) modified for non-convexities was used to solve the resulting problem just for testing the idea that NUDM formulation is superior over UDM when the number of time intervals, generated by UDM, is large. This was shown by a simple test example. The OA algorithm is intended for convex functions. In the non-convex cases the OA algorithm may miss the optimal solution. In [97] an alternative NUDM formulation for batch process scheduling was given and solved using BHA instead of OA.

14.2.4 Non-Uniform Discrete-Time Algorithms

One of the possible ways of solving NUDM problems is to use stochastic approaches such as the Bayesian Heuristic Approach. The BHA framework allows us to employ a specific heuristic for a given class of problems.

In this chapter, we consider the same models as in [97] but present a simpler NUDM formulation and describe a global optimization algorithm based on BHA using MRP heuristics. The algorithm has the property that the solution time is growing slowly with the number of Boolean variables in contrast to Branch-and-Bound enumeration⁴. Another feature of the BHA-MRP algorithm is that the original mixed-integer non-linear optimization problem is transformed into a problem of calibration of randomized MRP heuristics. The calibration problem is continuous and is solved by the Bayesian global optimization methods (see [100]).

The computational experiments, described in the next chapter suggest, that using BHA and the Non-Uniform models one significantly outperforms the exact methods and the Uniform Models, when the number of Boolean variables needed by the UDM is large.

⁴B&B is known to be an exponential time algorithm for NP -complete problems (see Section 2.2.5).

In the following two chapters we consider some well known global optimization algorithms, namely the Simulated Annealing (SA) and the Genetic Algorithm (GA) in the framework of BHA.

In the next chapter while applying SA we describe the batch/semi-continuous scheduling problem as a Mixed Integer Bilinear Programming (MIBLP) one (see Chapter 13). In this case and also in the case of MRP heuristics, constraints are represented as penalty functions. The difference is that in the MRP case, the penalty function just helps to keep the decisions feasible. In the SA case, the penalty function is a part of heuristic. The BH algorithm, based on the SA randomization of MIBLP penalty function heuristics and the computational results of both the SA and MRP cases, are described in the next chapter. We illustrate the performance of the algorithm, compare the the results with the MRP case, and draw some conclusions.

Considering GA, the batch scheduling is represented as an Mixed Integer Linear (MILP) problem and constraint feasibility is obtained by choosing appropriate mutation and cross-over techniques. The algorithm, based on the GA techniques using the Mixed Integer Linear Programming representation, is described in Chapter 16.

An illustrative example of a batch process model, based on a Non-Uniform Discrete-Time approach, is presented in the next section. In section 14.4 the discussion of the BH approach is given. The algorithm, based on the "polynomial randomization" (see Subsection 11.5.1) of MRP heuristic, is described in Section 14.5.

14.3 ILLUSTRATIVE EXAMPLE

To explain NUDM consider a batch processes example BATCH1 [135]. In the next chapter the NUDM model is described in MIBLP terms. In Chapter 16 the MIBLP description is reduced to the MILP form. The *state-task* network and relevant data for the illustrative example are shown in Table 14.1. The data from [135] was slightly modified adapting to our needs. The objective is to maximize profit at the end of 4 hours.

The uniform discrete-time approach generates 40 intervals of size 0.1. Thus 82 Boolean variables are needed in the case when only two Boolean variables

Table 14.1 Data of Illustrative Example

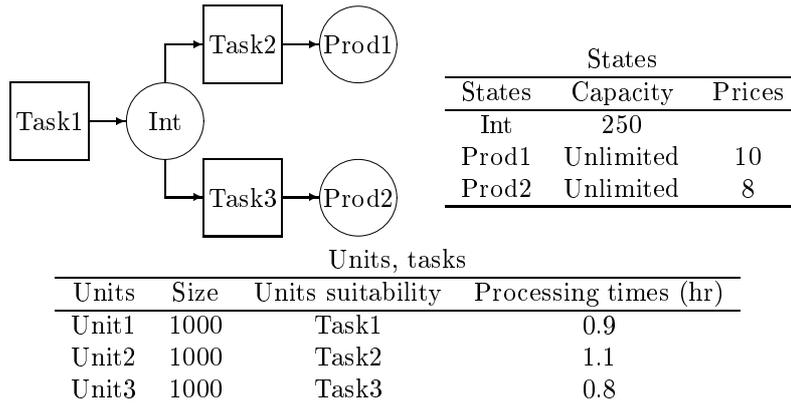
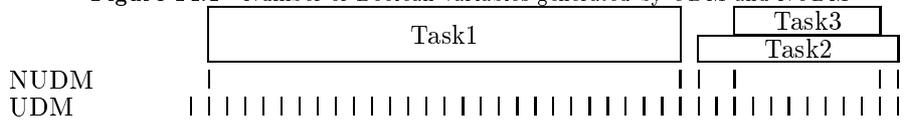


Figure 14.1 Number of Boolean variables generated by UDM and NUDM



represent each interval. The nonuniform discrete-time approach requires for only 12 Boolean variables (see Figure 14.1).

Task processing times are as follows:

$$\begin{aligned}
 i = Task1 & \quad \tau_i = 0.9 \\
 i = Task2 & \quad \tau_i = 1.1 \\
 i = Task3 & \quad \tau_i = 0.8
 \end{aligned}$$

The example is described by these parameters:

S_{so} is amount of material stored in state s at time t_o ;

N_{io} is number of units i available at time t_o ;

- B_i is amount of material processed by task i ;
- t_i^S is start time of task i ;
- t_i^F is finish time of task i ;
- $W_{io}^S = 1$, if and only if $t_i^S = t_o$;
- $W_{io}^F = 1$, if and only if $t_i^F = t_o$;
- τ_i is task i processing time;
- c_s is the unit price associated with material in state s .

The independent decision variables are batch sizes B_i and starting times t_i^S of each task i . If these are fixed, then the finish times $t_i^F = t_i^S + \tau_i$ for each task i are readily calculated and so the event set $X = \{t_i^S\} \cup \{t_i^F\}$ is defined. The Boolean variables W_{io}^S (W_{io}^F) which have the value of one only if $t_o = t_i^S$ or $t_o = t_i^F$ are defined by the decision variables. These variables are needed writing model equations such as material balances, capacity and allocation constraints.

Material balances show the amount of material accumulated in state s at given time t_o :

State	Int	Prod1	Prod2
S_{so}	$\sum_{k=1}^o \{ \sum_{i=Task1} B_i W_{ik}^F - \sum_{i=Task2, Task3} B_i W_{ik}^S \}$	$\sum_{k=1}^o \sum_{i=Task2} B_i W_{ik}^F$	$\sum_{k=1}^o \sum_{i=Task3} B_i W_{ik}^F$

Demands and feeds are readily incorporated. The material balances introduce non-linearity and multi-modality into the model.

The amount of material stored in a state s must not exceed the maximum storage capacity for this state, it means that

$$0 \leq S_{so} \leq \begin{cases} 5000 & \text{if } s = Int \\ \infty & \text{if } s = Prod1 \\ \infty & \text{if } s = Prod2. \end{cases}$$

An idle item of equipment can start at most one task. If the item does start performing a given task, then it cannot start any other task as long as the current one is not finished, i.e., the operation is non-preemptive.

We express the allocation constraints in this way:

$$N_{io} = \sum_{k=1}^o \left\{ \sum_i W_{io}^F - \sum_i W_{io}^S \right\},$$

where N_{io} represents unit i ability to process a task. Unit i is taken from the "pool" of units when it begins processing some task and so is unavailable to process other tasks. It is returned to the same pool when it finishes the processing of this task and thus is available to process other tasks. The temporary unavailability of equipment due to the maintenance or breakdown is modeled in the same way.

The objective to be maximized, is the value of products:

$$\sum_s c_s S_{s,|X|}, c_s = \begin{cases} 10 & \text{if } s = \textit{Prod1} \\ 8 & \text{if } s = \textit{Prod2}. \end{cases}$$

We maximize the profit by varying the decision variables B_i and t_i^S while satisfying the constraints.

14.4 APPLYING BHA

14.4.1 Randomizing Heuristics

In general, a decision process consists of a number of steps (see Chapter 11). In batch scheduling problems these steps are: selecting a task, selecting a suitable equipment unit processing the task, and the amount of material to be processed by this task. At each step we have to select an object from some decision set, for example, select a task from the set of tasks producing the necessary product.

A heuristic is a set of rules used in performing a step. Suppose that some weights are assigned to all objects in the decision set. It may be helpful to think of heuristic rules as a rule of selecting an object with the maximal "weight". The BHA idea is to randomize these rules (see Section 11.4). Instead of selecting an object with the maximal "weight" we select an object with some probability which is a function of its weight. This function is called a randomized heuristic.

Here a similarity is noticeable with Genetic Algorithms (GA) in that an object is included into a new population with a probability that is increasing with the

weight of the object. The difference is that BHA optimizes the relation of this probability to the weight (see Chapter 16), while in the traditional GA this relation is fixed (see [62]).

The goal of randomization is to increase the efficiency of the search (see Section 3.3). Thus, instead of using the heuristic directly, we select a decision with some probability, that is a parametric function of the given heuristic (see Section 11.5). The randomized heuristic function can be adapted to a given class of optimization problem by optimizing its parameters repeatedly applying optimization procedures (see Section 11.6). Thus the original optimization problem is replaced by an auxiliary problem of continuous stochastic optimization which is solved by the Bayesian methods of global optimization [100]. Using the Bayesian terms, this implies some a priori distribution to be assumed on a set of randomized heuristics (see Section 11.9).

The simplest example is the knapsack problem (see Section 12.1). A commonly used heuristic is selecting an object with a maximal specific value (the ratio of the value and the weight of the object). The randomized heuristic selects an object with a probability that is a parameterized function of its specific value. By varying parameters of this function we are looking for the best parameters while preserving the positive trends of this heuristic.

14.4.2 Penalty Function

The feasible region of a the model defined in Section 14.3 is quite complex. However, Bayesian global optimization methods, as usual, are designed for a hyper-rectangle. Thus some tool is needed to extend the feasible region to this hyper-rectangle. The penalty function is a convenient way to do this. Since there are Boolean and continuous variables, we need two components for a penalty function: one for violation of Boolean constraints and one for violation of continuous constraints.

14.5 BAYESIAN APPROACH USING MRP HEURISTICS

Once the randomized heuristic function is defined in BHA, we optimize its parameters using Bayesian algorithms, see [100] and Chapter 4. The objective is the highest profit reached after a fixed number of repetitions. Using BHA

one obtains the optimal randomization parameters as a "byproduct". These parameters may be used while solving other similar problems.

In this chapter, a parametric second order polynomial randomization (see Subsection 11.5.1) of a Material Requirements Planning heuristics is employed. The feasible region of this heuristic is complicated, therefore a penalty function is used to extend the feasible region to a hyper-rectangle. Using MRP as heuristics we penalize just the excess amount of material in the given state. All the other constraints, Boolean and continuous, are not violated while applying the MRP algorithm. Thus, only one penalty parameter is needed for the continuous constraints which are set to the value of 500. That is fairly much as compared with the value of products but it is comparable to the profit.

Using MRP we try to schedule tasks as near to the deadline as possible. Thus we define the MRP heuristics as the function of the distance D_i between the deadline and the finish time of a task i .

If d_i is a decision to schedule a task i (or to select an equipment unit), then $h(d_i)$ is the heuristic. The exponential function of the distance $h(d_i) = \exp\{-D_i\}$ has given the best results. The function $r(x, h(d_i))$ is a randomized heuristic function, i.e., with this probability we select the decision d_i . The function r used in this chapter has the following form:

$$r(x, h(d_i)) = x_0 a_0 + x_1 a_1 h(d_i) + x_2 a_2 h^2(d_i),$$

where

$$x_0 + x_1 + x_2 = 1, \quad a_0 = \frac{1}{M}, \quad a_1 = \frac{1}{\sum_{i=1}^M h(d_i)}, \quad a_2 = \frac{1}{\sum_{i=1}^M h^2(d_i)}.$$

Here M is the number of possible decisions and x is the parameter set by the Bayesian method. An exponential heuristic function is applied instead of a linear one so that the probability to select product with late due date would be very small. As we see the products with smallest due dates are chosen with highest probability, thus we preserve the useful trend of the the MRP heuristic while making it random. In a similar manner we deal with other rules.

The generated schedule may be infeasible because of violation of the state capacity constraints. It is possible that the state capacity is exceeded or we have a negative amount of material in a given state. These situations are handled by penalizing the variations from the minimal and the maximal state capacity

values and adding this penalty to the objective function. The objective function is readily evaluated by calculating the storage and utility costs, the profit gained by satisfying demands less the raw material costs.

14.5.1 Heuristic Priority Rules

We next discuss the application of the Material Requirement Planning Heuristic (MRP). MRP is an inventory management and production planning technique which, given a delivery schedule for a final products, determines the initiating times for all raw materials orders, for the production runs needed to prepare all required intermediate products, as well as the starting times for the production of the final product itself. Given a delivery time for a product shipment, each branch of the product processing tree is traced from the product in question and each component requirement is calculated. If the inventory is inadequate, then a production order is issued for that component. The tracing of each branch of the processing tree continues until all raw materials requirements have either been met or ordered.

For scheduling problems relevant to the chemical industry we have to extend this heuristic to handle unit and task assignment, batch size determination, and other features. For the purpose of this book we will employ three simple and natural heuristic rules:

- *Product selection rule.* The production run of each product in a batch and continuous processing system has a due date. One begin with a product that has the earliest due date. The rationale for this is that later due dates are less certain. When the production of this product is scheduled one recursively schedules the production of intermediates required for producing it.
- *Equipment item selection rule.* To schedule the production of a given product or an intermediate one has to decide a task and to select an equipment unit to process this task. When there are several tasks producing the same product, we randomly assign the quantity of a product to be produced. We select an equipment item that is free during the time interval, closest to the deadline.
- *Task start selection rule.* Usually this time interval is longer than the task processing time. Thus we start a given task so that it ends exactly at the end of this interval.

Using MRP heuristics BHA algorithm is represented by the following steps:

- Step 1. Fix parameters x using the global Bayesian method [100];
- Step 2. Evaluate the schedule for these parameters;
- Step 3. If the number of iterations is greater than the maximal value, then stop;
- Step 4. Go to step 1.

We comment the first two steps:

- Step 1. The parameter vector $x = (x_0, x_1, x_2)$ is controlled by the global Bayesian method [100]. Note, that any method generating a positive vector x provides the convergence with probability one, see Theorem 3.2.1. We use namely BA for both the theoretical and the practical reasons.

Theoretically BHA is interesting because it unites the Bayesian approach and Heuristic programming by defining an a priori distribution on a family of heuristics.

The practical reason is that Bayesian methods are designed for the global stochastic optimization so we may expect good results optimizing a multimodal and stochastic auxiliary function $f(x)$;

- Step 2. The schedule is formed by using randomized MRP heuristics at a fixed parameter vector x . The value of this schedule is the profit (for example, the value of products). It is possible that the generated schedule is infeasible, i.e., the capacity for some states is exceeded or some states contain negative amount of material. Then we penalize it;

The results are given in Table 15.3 (see the next chapter) which includes the results of both the BHA algorithms: one using the MRP heuristics with polynomial randomization, and the other one using the randomization of simulated annealing type (see expression 11.23).

15

BATCH PROCESS SCHEDULING USING SIMULATED ANNEALING

15.1 INTRODUCTION

The problem is formulated in the Mixed Integer Bilinear Program (MIBLP) form (see Chapter 13) as in the previous chapter. A difference is that here we optimize the randomization procedure of Simulated Annealing type using the MIBLP penalty function as a part of heuristics. Another difference is that we describe the complete batch scheduling problem, not just illustrative example as in Section 14.3 of the last chapter.

Computational comparisons, applying BHA to several NUDM test problems and both the MRP and the SA heuristics, are made against a Uniform Discrete-time Model.

15.2 OUTLINE

A Non-Uniform Discrete-time Model is considered in this chapter. This model is not as general as the Non-uniform Continuous-time one [160] because the assumptions of [160] are less restrictive. However, the constraints, generated using the *state-task* network in this chapter, are the same as those generated using the *resource-task* in [160].

A model similar to [105] is considered here.. We make it simpler under the same assumptions to create a more efficient algorithm using the generic heuristics calibration idea [94].

15.3 MODEL DESCRIPTION

Each task is assumed to occur once in a given equipment unit. This is not an unduly restrictive assumption since one can readily estimate the number of times a task can be executed and thus create the corresponding number of identical tasks. In case a task is not executed at all, the corresponding batch size is set to zero. It is also assumed that the output from a task to all its output states occurs at the same time. To model separation processes where this is not the case we create additional tasks with different processing times corresponding to different output times. States which connect these tasks are then required to be of zero wait type.

The key variable in the model is the timing of events. We denote an event by X_o and the time of occurrence of this event by t_o . The set of all events is denoted by X .

15.3.1 Constraints

The constraints include:

- material balances;
- limitations on the capacities of units and storage states; and
- allocation of equipment units to tasks.

These constraints occur in all the batch scheduling problems. In addition, there could be other factors, too. For example, temporary unavailability of equipment during the time horizon, due dates on the delivery of products to customers, the late arrival of raw materials, a limited availability of utilities and manpower, equipment cleaning requirements, etc.

Material Balances

It is often necessary (e.g., due to contractual obligations) to deliver to customers certain agreed quantities $D_{sl}, l = 1, \dots, D_s$ of material in product state s at various times t_{sl}^D . It may be necessary (e.g., due to a limited availability of local storage capacity), or desirable (e.g., due to price variations) to receive quantities $R_{sl}, l = 1, \dots, R_s$ of raw materials in feed state s at times t_{sl}^R rather than having all the required feed-stocks stored locally at the beginning of processing.

The corresponding material balances are expressed as follows.

$$S_{so} = S_{s,o-1} - D_{slo} + R_{slo} - \sum_{i \in T_s} \rho_{is} \sum_{j \in J_i} B_{ij} W_{ij_o}^S + \sum_{i \in \overline{T_s}} \overline{\rho}_{is} \sum_{j \in J_i} B_{ij} W_{ij_o}^F, \forall s, o \tag{15.1}$$

where

- S_{so} is the amount of material stored in state s at time t_o ;
- D_{slo} is the amount of material delivered at time t_o ;
- R_{slo} is the amount of material received at time t_o ;
- B_{ij} is the amount of material processed by task i on unit j ;
- J_i is the set of units which can process task i ;
- t_{ij}^S is the start time of task i processing on unit j ;
- $W_{ij_o}^S = 1$ if $t_{ij}^S = t_o$;
- τ_{ij} is the task i processing time on unit j ;
- t_{ij}^F is the finish time of task i processing on unit j : $t_{ij}^F = t_{ij}^S + \tau_{ij}$;
- $W_{ij_o}^F = 1$ if $t_{ij}^F = t_o$;
- T_s is the set of tasks receiving material from state s ;
- S_i is the set of states which feed task i ;
- ρ_{is} is the proportion of input of task i from state $s \in S_i$: $\sum_{s \in S_i} \rho_{is} = 1$;
- $\overline{T_s}$ is the set of tasks producing material of state s ;
- $\overline{S_i}$ is the set of states which receive from task i ;
- $\overline{\rho}_{is}$ is the proportion of output of task i to state $s \in \overline{S_i}$: $\sum_{s \in \overline{S_i}} \overline{\rho}_{is} = 1$.

This constraint simply states that the net increase $S_{so} - S_{s,o-1}$ in the amount of material stored in a state s at time t_o is given by the difference of the amounts produced and used in this state.

A similar notation list is given while explaining the illustrative example in Section 14.3. We did repeat that list here, including some extension, for reading convenience ¹.

Capacity constraints

The amount of material stored in a state s must not at any time exceed the maximum storage capacity for this state:

$$0 \leq S_{so} \leq S_s^{max}, \forall s, o. \quad (15.2)$$

The amount of material that starts undergoing task i in unit j at time t_o is bounded by minimum and maximum capacities of that unit:

$$B_{ij}^{min} \leq B_{ij} \leq B_{ij}^{max}, \forall i, j \in J_i. \quad (15.3)$$

Allocation constraints

At any time, an idle item of equipment can start at most only one task. If the item does start performing a given task, then it cannot start any other task as long as the current one is finished, i.e., the operation is non-preemptive.

The allocation constraints are expressed as:

$$N_{jo} = N_{j,o-1} - \sum_{i \in I_j} W_{ijo}^S + \sum_{i \in I_j} W_{ijo}^F, \forall j, o \quad (15.4)$$

where

N_{jo} is the number of units j available at time t_o ;

I_j is the set of tasks which can be performed by unit j .

¹For the same reason we shall repeat some of these expressions once more while defining the batch scheduling process in the MILP form (see Section 16.2).

15.3.2 Objective function

The criterion is the maximization of profit. We express the profit as:

$$\text{profit} = \text{value of products} - \text{cost of feed} - \text{cost of storage} - \text{cost of utilities} \quad (15.5)$$

$$\text{value of products} \equiv \sum_s c_s \left(\sum_{l=1}^{D_s} D_{sl} + S_{s,|X|} \right) \quad (15.6)$$

$$\text{cost of feed} \equiv \sum_s c_s (S_{s0} + \sum_{l=1}^{R_s} R_{sl}) \quad (15.7)$$

$$\text{cost of storage} \equiv \sum_s c_s^M (S_{s,|X|} \text{horizon} + \sum_{o=1}^{|X|} t_o (S_{s,o-1} - S_{so})) \quad (15.8)$$

$$\text{cost of utilities} \equiv \sum_u \sum_{i \in I_u} \sum_{j \in J_i} \tau_{ij} (\alpha_{uij} \sum_{o=1}^{|X|} W_{ijo}^S + \beta_{uij} B_{ij}), \quad (15.9)$$

where

- c_s is the unit price associated with material in state s ;
- c_s^M is the running cost of storing a unit amount of material in state s ;
- I_u is the set of tasks using utility u ;
- α_{uij} is a constant demand factor;
- β_{uij} is a variable demand factor.

15.4 FORMULATION EXAMPLE

Denote the parameters as t_{sl}^D and t_{sl}^R .

the decision variables as B_{ij} and t_{ij}^S .

the equations as

$$t_{ij}^F = \begin{cases} t_{ij}^S + \tau_{ij} & \text{if } B_{ij} > 0 \\ t_{ij}^S & \text{otherwise,} \end{cases} \quad (15.10)$$

$$X = \{t_{ij}^S\} \cup \{t_{ij}^F\} \cup \{t_{sl}^D\} \cup \{t_{sl}^R\} \quad (15.11)$$

$$D_{slo} = \begin{cases} D_{sl} & \text{if } t_o = t_{sl}^D \\ 0 & \text{otherwise,} \end{cases} \quad (15.12)$$

$$R_{slo} = \begin{cases} R_{sl} & \text{if } t_o = t_{sl}^R \\ 0 & \text{otherwise,} \end{cases} \quad (15.13)$$

$$W_{ijo}^S = \begin{cases} 1 & \text{if } t_o = t_{ij}^S \text{ and } B_{ij} > 0 \\ 0 & \text{otherwise,} \end{cases} \quad (15.14)$$

$$W_{ijo}^F = \begin{cases} 1 & \text{if } t_o = t_{ij}^F \text{ and } B_{ij} > 0 \\ 0 & \text{otherwise.} \end{cases} \quad (15.15)$$

The material balances are defined by expression (15.1), the state capacity is determined by inequalities (15.2), the batch size limits are described by inequalities (15.3), and the allocation constraints are given by inequalities (15.4).

Note, that:

- a task can be assigned to different equipment units;
- expression (15.10) enforces the requirement that operations be non-preemptive;
- expressions (15.12-15.13) enforce that sales and purchases take place at required times;
- the Boolean variables W are not decision variables.

15.4.1 Reducing Discrete Optimization to Continuous One

In some cases it is convenient to reduce the discrete optimization problem to the global optimization of continuous variables. One may do this by including logical conditions and discrete parameters into some algorithmically defined objective and constraints.

The model, described before, defines a mixed integer non-linear non-convex problem. Both the objective function and the constraints are non-linear. A

specific difficulty is the presence of non-linear equality constraints. This is the price one has to pay for reducing the number of independent variables.

We try to avoid infeasible solutions by penalizing the constraint violations. Thus the original mixed integer nonlinear optimization problem is reduced to a continuous global optimization one. We use BHA techniques (see [94]) to obtain an approximate solution of this problem. Using the BHA a positive probability density of any feasible point is provided. Thus the convergence conditions (see Theorem 3.2.1) are satisfied.

We optimize the parameters of the randomized heuristics using the global optimization method [100]. The penalized profit function is considered as heuristics in this approach. Simulated Annealing is used as a randomization technique and the initial "temperature" as the parameter to be optimized.

15.4.2 Penalty Function

The penalty function $L_n(B_{ij}, t_{ij}^S)$ is used to extend the feasible region to a hyper-rectangle. We define two components of the penalty function:

$$L_n(B_{ij}, t_{ij}^S) = P_n^b(B_{ij}, t_{ij}^S) + P_n^c(B_{ij}, t_{ij}^S) \tag{15.16}$$

Here $P_n^b(B_{ij}, t_{ij}^S)$ is a penalty for the violation of Boolean constraints in iteration n , and $P_n^c(B_{ij}, t_{ij}^S)$ is a penalty for violation of continuous constraints in iteration n .

The right choice of penalty parameters r_n^b and r_n^c is an important problem when using penalty functions. If we set too large values of those parameters, then the optimization problem may degenerate into the search for the "nearest" feasible decision. If we set too small values for r_n , then we may violate the constraints. Some compromise may be reached by increasing penalty parameters after each iteration.

The Boolean penalty parameter r_n^b must be much greater than the continuous one r_n^c . The reason is that some violation of continuous constraints is permissible, if the constraints are defined by economical considerations. If not, then one may include some "safety margin" while defining "strict" continuous constraints. Unfortunately, we cannot do such things in the case of Boolean constraints. We may ignore the strict nature of Boolean constraints at the initial stages of global optimization, when the optimum is still far away, but have to observe those constraints exactly when approaching the global optimum.

Thus the penalty function is as follows:

$$\begin{aligned}
 L_n(B_{ij}, t_{ij}^S, r^b, r^c) = & \quad (15.17) \\
 & \sum_{o=1}^{|X|} \sum_j r^b \max\{0, N_{jo} - 1, -N_{jo}\} + \\
 & \sum_i \sum_{j \in J_i} r^c \max\{0, B_{ij} - B_{ij}^{max}, B_{ij}^{min} - B_{ij}\} + \\
 & \sum_{o=1}^{|X|} r^c \max\{0, t_o - horizon, -t_o\} + \\
 & \sum_{o=1}^{|X|} \sum_s r^c \max\{0, S_{so} - S_s^{max}, -S_{so}\}.
 \end{aligned}$$

15.4.3 Penalized Profit as Heuristics

A direct way of optimizing a penalized profit function

$$L(B_{ij}, t_{ij}^S) = profit - L_n(B_{ij}, t_{ij}^S) \quad (15.18)$$

is by the usual continuous global optimization techniques. Unfortunately, the techniques are designed for the case when the penalty parameters r_n do not depend on the iteration number.

If the number of variables is not large, one may apply the B&B techniques. However, if the number of variables is large, then the Bayesian Heuristics Approach seems a good alternative.

We apply BHA by fixing some initial decision (B_{ij}, t_{ij}^S) (not necessarily feasible), and then improving it by permutations. The "best" permutations are selected by randomized heuristics (regarding the penalized profit (15.18) as heuristics).

A permutation of the current decision (B_{ij}, t_{ij}^S) may be performed sequentially by adding a Gaussian $(0, \sigma)$ random variable to each component of B_{ij} and t_{ij}^S one by one. A more sophisticated permutation would be to add a Gaussian random vector to several components at a time. In the latter case, we need some expert knowledge for choosing the right combination of components.

15.4.4 Simulated Annealing as Randomized Heuristics

There are many ways of randomizing heuristics. The simplest one is Simulated Annealing. In this case, one has to optimize only one randomization parameter, namely, the initial temperature.

Altho there are a number of publications which use simulated annealing for different scheduling problems [145, 43, 26, 120], no regular optimization of the simulated annealing parameters was performed.

We regard Simulated Annealing as the first step in considering the Bayesian Heuristic Approach to the optimization of batch scheduling. Different approaches will be investigated and compared: one of them is the Genetic Algorithm (see Chapter 16).

15.4.5 BHA Algorithm

We describe the algorithm in terms of the following steps.

- Step 1. Select not necessarily a feasible initial point (B_{ij}, t_{ij}^S) . Denote it by $(B_{ij}^{best}, t_{ij}^{Sbest})$, if it is feasible.
- Step 2. Set *global iteration* to 1.
- Step 3. Select T_0 with a global Bayesian method.
- Step 4. Set *iteration* to 1, r^b to r_0^b , and r^c to r_0^c .
- Step 5. Set the initial point for simulated annealing to be $(B_{ij}^{best}, t_{ij}^{Sbest})$.
- Step 6. Calculate y to be equal to $-profit + Penalty(B_{ij}, t_{ij}^S)$. Increase r^b by r_0^b , and r^c by r_0^c for each fixed number iterations.
- Step 7. Perturb the point to get a new point $(B_{ij}^{new}, t_{ij}^{Snew})$.
- Step 8. Calculate y^{new} to be equal to $-profit + Penalty(B_{ij}^{new}, t_{ij}^{Snew})$.
- Step 9. Go to the new point, if y^{new} is less than y or if y^{new} is greater than y with probability $\exp \frac{y - y^{new}}{T_0 / \ln(1 + iteration)}$.

- Step 10. Select the best point $(B_{ij}^{best}, t_{ij}^{Sbest})$ which is feasible and gives a greater *profit*.
- Step 11. Increase *iteration* by 1.
- Step 12. Go to Step 6, if *iteration* is less than *max_iterations*.
- Step 13. Increase *global_iteration* by 1.
- Step 14. Go to Step 3, if *global_iteration* is less than *max_global_iterations*.
- Step 15. Fix t_{ij}^S to t_{ij}^{Sbest} , create corresponding LP and solve it.

The traditional logarithmic cooling schedule of simulated annealing is used in this BHA algorithm. Using BHA this schedule is not so important, because in the Bayesian framework the convergence conditions are satisfied just by keeping the probability density positive for all feasible points (see Theorem 3.2.1). Therefore BHA satisfies weaker convergence condition (3.5). SA satisfies stronger convergence condition (3.6), if the cooling schedule is right, and if the initial temperature is high enough ².

15.5 COMPUTING RESULTS

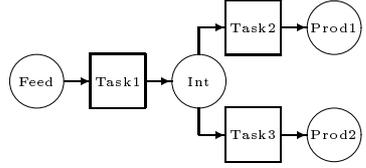
Here we discuss the computing results of both the previous and this chapter. We compare various BHA algorithms with the truncated solutions of the MILP uniform discrete-time model using B&B [163]. The results are reported for different test examples, both the batch and the continuous ones.

The results are summarized in Table 15.3. There are several lines for some examples. The successive lines relate to stopping *B&B* enumeration after 10,000, 20,000, and 30,000 nodes, respectively. A different number of nodes is reflected in the table as different solution time. It is surprising to see that the "optimal" *B&B* profit remains the same in all the three lines. The "optimal" BHA profit increases with the number of observations reflected as the solution time. Denote by BHA-1 and BHA-2 two versions of BHA using MRP heuristics. Denote by BHA-3 the BHA version using SA and the MIBLP penalty function as heuristics.

The two lines marked 'uni-mod' correspond to an unmodified version of the batch scheduling problem where the uniform discrete-time interval length (1

²As usual, it is not clear how to test the last condition.

Table 15.1 Data used for example BATCH1



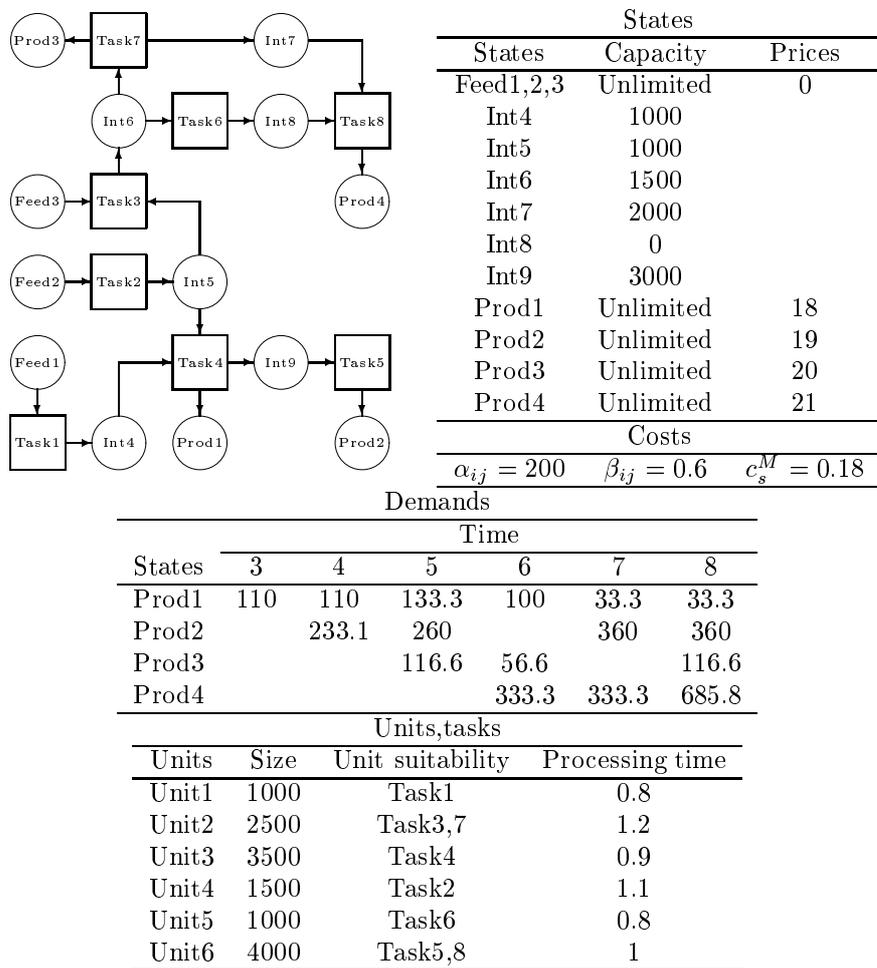
States		
States	Capacity	Prices
Feed	Unlimited	5
Int	5000	
Prod1	Unlimited	10
Prod2	Unlimited	8

Costs		
$\alpha_{ij} = 200$	$\beta_{ij} = 0.6$	$c_s^M = 0.18$

Demands						
States	Time					
	4	6	7	10	11	12
Prod1	200		300	400	100	
Prod2	50	150		200		100

Units, tasks			
Units	Size	Unit suitability	Processing time
Unit1	1500	Task1	0.9
Unit2	1000	Task2	1.1
Unit3	1000	Task3	0.8

Table 15.2 Data used for example BATCH4



for both examples) is comparable to the scheduling horizon. As seen, NUDM performs better even in this case. In theory, in such a case, UDM has to outperform NUDM. A possible explanation is that the MRP heuristics suit these examples well.

The computational experiments were performed using a HP 9000/755 workstation. For some batch processes (BATCH3, BATCH4, BATCH8), BHA gives only suboptimal solutions. For both the continuous processes CONT1 and CONT2, BHA gives better results than UDM.

NUDM works much better than UDM for problems with sequence dependent tasks (CONT1, CONT2). This is due to the fact that NUDM handles sequence-dependent tasks in a more efficient way.

Examples BATCH1 and BATCH4 are from [135]. The *state-task* networks for these problems are shown in Figures 15.1 and 15.2. The problem parameters are given in Tables 15.1 and 15.2. The examples were modified so that the uniform discrete-time interval length (0.1 for both examples) is much smaller than the scheduling horizon (12 for BATCH1 and 8 for BATCH4). This is achieved by slightly perturbing the processing time from the original integer value.

Processes BATCH6, BATCH7, and BATCH8 differ by the increasing number of sequence dependent tasks. That explains why the solution time by B&B grows faster as compared with that by BHA. The attempt to solve BATCH8 exactly by B&B was not successful, the solution time was too great (see the mark * in Table 15.3). The B&B solution of example CONT1 was terminated after 20000 nodes for the same reason (see the mark ** in Table 15.3).

We compared the three versions of BHA algorithm (BHA-1, BHA-2, and BHA-3) with the MILP uniform discrete-time model (denoted as B&B) which was solved by the Branch-and-Bound (see [163]). Two BHA versions: BHA-1 and BHA-2 represent the MRP heuristics using a different number of iterations. The version BHA-3 implements BHA by optimizing the SA initial "temperature" using a "penalized profit" of the MIBLP model as heuristics for the Batch/Semi-Continuous Scheduling Problem. Detailed numerical data for these examples is available electronically from the rcsplib account (*rcsplib@ecn.purdue.edu*) in the form of an RCSPec language files.

Clearly BHA is more efficient compared with the truncated B&B in UDM, if the discrete-time interval is not large. The results also confirm that in the NUDM formulation the number of variables and constraints does not depend

Table 15.3 BHA and B&B comparison

Problem	Alg.	Profit	CPU time	Numb. Boolean var.	Numb. cont. var.	Numb. constr.
BATCH1	B&B	2744.5	584.9	305	860	1073
		same	1217.5	same	same	same
		same	1901.2	same	same	same
	BHA-1	2696	0.03	18		
		2939	0.04	14		
	BHA-3	3007	6.9	384	376	744
		3079	85.4	same	same	same
3079		1396.5	same	same	same	
uni-mod	B&B	3230	6			
		3230	11.21	36	101	88
	BHA-1	3230	0.03	24		
	BHA-2	3230	0.9			
	BHA-3	3230	19.34	384	376	744
BATCH3	B&B	105756	8.6			
	BHA-2	105702	1.5			
BATCH4	B&B	60224	944.8	488	1553	1823
		same	1919.7	same	same	same
		same	2944.5	same	same	same
	BHA-3	60284	103.0	2240	1776	3976
		60306.2	163.2	2240	same	same
		60311.7	1276.8	2240	same	same
	BHA-1	60459.2	21.28	54		
		60468.3	63.36	54		
		60533.5	1.7			
	uni-mod	B&B	60533.5	1.86	56	176
60533.5			0.19	50		
BHA-2		60471.6	3.6			
BHA-3		60474.7	309.7	2240	1776	3976
BATCH5	B&B	1400	66			
	BHA-2	1400	30			
BATCH6	B&B	4724	0.6			
	BHA-2	4724	0.05			
BATCH7	B&B	8122	188.5			
	BHA-2	8122	1.15			
BATCH8	B&B	1205	*			
	BHA-2	11996	13.26			
CONT1	B&B	20879	**			
	BHA-2	20992	13.6			
CONT2	B&B	22800	824.6			
	BHA-2	23346	51.6			

on the discrete-time interval length. In the UDM formulation the corresponding relation is exponential.

15.6 ANALYSIS OF RESULTS

It is important to choose the right scheduling approach. In the case of a homogeneous processing time, UDM is adequate. In the case of a non-homogeneous processing time, NUDM works better.

The proper schedule can lead to significant improvement in processing efficiency. On the other hand, obtaining the optimal schedule can require a large amount of computational time. Thus we need “good” engineering judgment to find a compromise between the cost and the benefit of schedule improvement. The Bayesian Heuristic Approach, combined with the Non-Uniform Discrete-time Model is a convenient tool for achieving that compromise.

In the previous chapter the polynomial type randomization of specific MRP heuristics was optimized. In this chapter, the initial “temperature” of SA type randomization was optimized using the MIBLP penalty function as heuristics. While the computational evidence is only preliminary, the results are encouraging. In theory, a polynomial type randomization should lead to the improved performance since one has more parameters to be tuned up.

The Bayesian Heuristic Approach is stochastic by definition. Therefore the application of this approach to optimize batch processes with random parameters (see Section 11.10) is straightforward. It is well known that an a priori distribution is a convenient way to represent uncertainties in the model, see [27]. This way BHA includes uncertain parameters, too. The present formulation may be easily expanded to modeling continuous processes, a temporary unavailability of equipment, and changeovers.

It is not the goal of this book to compare different batch scheduling models, such as UDM and NUDM. However, we may conclude, as a “byproduct”, that NUDM yields smaller optimization problems without sacrificing accuracy.

16

GENETIC ALGORITHMS FOR BATCH PROCESS SCHEDULING USING BHA AND MILP FORMULATION

16.1 INTRODUCTION

So far we have described two ways of applying BHA to the Batch Scheduling Problem. In Chapter 14 specific MRP heuristics were used. In Chapter 15 the SA "initial temperature" parameter was optimized while applying the MIBLP penalized profit function as a heuristic.

In this chapter we reduce MIBLP to Mixed Integer Linear Programming (MILP) and optimize the parameters of the Genetic Algorithm (GA) which is considered as a randomized heuristic.

The formulation of batch scheduling in the "MILP way" involves a much larger system of equations and inequalities. An advantage is that we see how to extend the results to any other MILP problem.

In the last section of this chapter the aggregation is dealt with.

16.2 EQUALITIES AND INEQUALITIES

The following expressions look like the expressions (15.1)-(15.9) in the previous chapter. The reason is that in the MILP we use similar notation and definitions as in the MIBLP. However we need more equations and inequalities to define the Batch Scheduling problem using the MILP form

$$S_{so} = S_{s,o-1} - \sum_l D_{slo} + \sum_l R_{slo}$$

$$- \sum_i \rho_{is} \sum_j B_{ijo}^S + \sum_i \bar{\rho}_{is} \sum_j B_{ijo}^F \quad (16.1)$$

$$N_{jo} = N_{j,o-1} - \sum_i W_{ijo}^S + \sum_i W_{ijo}^F. \quad (16.2)$$

We split B_{ijo}^S , B_{ijo}^F , D_{ilo} , and R_{ilo} into two components each

$$B_{ijo}^S = B_{ij}^{min} W_{ijo}^S + b_{ijo}^S \quad (16.3)$$

$$B_{ijo}^F = B_{ij}^{min} W_{ijo}^F + b_{ijo}^F \quad (16.4)$$

$$D_{slo} = D_{sl}^{min} a_{slo}^D + d_{slo} \quad (16.5)$$

$$R_{slo} = R_{sl}^{min} a_{slo}^R + r_{slo}. \quad (16.6)$$

The sum $D_{so} = \sum_l D_{slo}$ denotes the delivery at the moment t_o , the sum $R_{so} = \sum_l R_{slo}$ denotes the feed at the moment t_o .

The Boolean indicator $a_{slo}^D = 1$ denotes the delivery of material D_{slo} at the moment t_o , and $a_{slo}^R = 1$ denotes the feed of material R_{slo} at the moment t_o . Here $i = 1, \dots, I$, $j = 1, \dots, J$, $o = 1, \dots, O$, $l = 1, \dots, L$.

Variables b_{ijo}^S , b_{ijo}^F and deliver/feed parameters d_{slo} , r_{slo} satisfy the inequalities

$$0 \leq b_{ijo}^S \leq B_{ij}^{max} - B_{ij}^{min} \quad (16.7)$$

$$0 \leq b_{ijo}^F \leq B_{ij}^{max} - B_{ij}^{min} \quad (16.8)$$

$$0 \leq d_{ilo} \leq D_i^{max} - D_i^{min} \quad (16.9)$$

$$0 \leq r_{ilo} \leq R_i^{max} - R_i^{min}. \quad (16.10)$$

The Boolean variables a_{slo}^D , a_{slo}^R are called "indicators" and the continuous variables d_{slo} , r_{slo} are called "delivery/feed parameters", because at this stage they are considered as fixed. We will regard them as variables later, when considering an aggregation of the batch scheduling problem.

It is convenient to select scales of B_{ij} such that $B_{ij}^{max} - B_{ij}^{min} = 1$. Variables W_{ijo} and indicators a_{ilo}^D , a_{ilo}^R are all Boolean

$$W_{ijo}^S = \{0, 1\}, W_{ijo}^F = \{0, 1\}, a_{ilo}^D = \{0, 1\}, a_{ilo}^R = \{0, 1\} \quad (16.11)$$

Variables W_{ijo} , B_{ijo} and indicators a_{ilo}^D , a_{ilo}^R satisfy equalities

$$\sum_{k>o} B_{ijk}^F = B_{ijo}^S \quad (16.12)$$

$$\sum_{k>o} W_{ijk}^F = W_{ijo}^S \quad (16.13)$$

$$\sum_l a_{ilo}^D = 1 \quad (16.14)$$

$$\sum_l a_{ilo}^R = 1 \quad (16.15)$$

and inequalities

$$1 - W_{ijo}^S \geq 1/\tau(t_o - t_{ij}^S), \quad 1 - W_{ijo}^S \geq 1/\tau(t_{ij}^S - t_o) \quad (16.16)$$

$$1 - W_{ijo}^F \geq 1/\tau(t_o - t_{ij}^F), \quad 1 - W_{ijo}^F \geq 1/\tau(t_{ij}^F - t_o) \quad (16.17)$$

$$1 - a_{ilo}^D \geq 1/\tau(t_o - t_{il}^D), \quad 1 - a_{ilo}^D \geq 1/\tau(t_{il}^D - t_o) \quad (16.18)$$

$$1 - a_{ilo}^R \geq 1/\tau(t_o - t_{il}^R), \quad 1 - a_{ilo}^R \geq 1/\tau(t_{il}^R - t_o). \quad (16.19)$$

Here t_{ij}^D , t_{ij}^R denotes "delivery/feed time", that was fixed.

$$\tau = aT, \quad a > 1, \quad T = \text{"horizon"} \quad (16.20)$$

$$0 \leq N_{jo} \leq 1 \quad (16.21)$$

$$\sum_o W_{ijo}^F = \sum_o W_{ijo}^S = W_{ij} \leq 1, \quad (16.22)$$

and

$$\sum_o W_{ijo}^S = W_{ij} = \begin{cases} 0, & \text{if } (i, j) \text{ is irrelevant} \\ 1, & \text{if } (i, j) \text{ is necessary} \\ q_{ij}, & \text{otherwise,} \end{cases} \quad (16.23)$$

where

$$q_{ij} = \{1, 0\} \quad (16.24)$$

$$\epsilon b_{ijo}^S \leq W_{ijo}^S \leq 1/\epsilon b_{ijo}^S \quad (16.25)$$

$$\epsilon b_{ijo}^F \leq W_{ijo}^F \leq 1/\epsilon b_{ijo}^F \quad (16.26)$$

$$\epsilon d_{ilo} \leq a_{ilo}^D \leq 1/\epsilon d_{ilo} \quad (16.27)$$

$$\epsilon r_{ilo} \leq a_{ilo}^R \leq 1/\epsilon r_{ilo} \quad (16.28)$$

$$t_{o+1} \geq t_o \quad (16.29)$$

$$t_{ij}^F = t_{ij}^S + \tau_{ij}. \quad (16.30)$$

Inequality (16.29) allows the multiple cases when $t_o + 1 = t_o$. That is not convenient defining material balance constraints. One may easily restrict the multiple cases replacing inequality (16.29) by the following one

$$t_{o+1} \geq t_o + \epsilon, \quad (16.31)$$

where $\epsilon > 0$ is a small number. Using inequalities (16.31) instead of (16.29) we consider a decision as an " ϵ -feasible" one, if the violation of any constraint does not exceed some " ϵ -limit".

In the case $W_{ijo}^S = 0$, all the values of t_o satisfy inequalities (16.17). We may avoid this uncertainty by including a small penalty term L_ϵ into the objective function (16.37):

$$\epsilon \sum_o t_o, \quad (16.32)$$

where $t_o \geq 0$. In such a case, all "irrelevant" t_o will be as small as possible depending on ϵ .

16.3 OBJECTIVE FUNCTION

An objective function is profit (16.37) defined by following four components:

$$\text{"value"} = L_v = \sum_s c_s \left(\sum_l D_{sl} + S_{ST} \right) \quad (16.33)$$

$$\text{"feed"} = L_F = \sum_s c_s \left(\sum_l R_{sl} + S_{s0} \right) \quad (16.34)$$

$$\text{"storage"} = L_S = T/R \sum_s c_s S_{so} \quad (16.35)$$

$$\text{"utilities"} = L_u = \sum_{u,i,j} (\alpha_{uij} \sum_o W_{ijo}^S + \beta B_{ijo}^S) \quad (16.36)$$

$$\text{"profit"} = L = L_v - L_F - L_S - L_u. \quad (16.37)$$

The problem is to maximize the profit

$$L^* = \max_{b^S, t^S} L \quad (16.38)$$

which is a function of real variables

$$b^S = (b_{ijo}^S), \quad t^S = (t_{ij}^S) \quad (16.39)$$

satisfying conditions (16.1)-(16.30) which depend on the real variables

$$b^F = (b_{ij}^F), t^F = (t_{ij}^F), t_o, B_{ij}^S, B_{ij}^F, S_{so} \quad (16.40)$$

and integer variables

$$W_{ij}^S, W_{ij}^F, N_{jo}, \quad (16.41)$$

where $i = 1, \dots, I, j = 1, \dots, J, o = 1, \dots, O$.

Here the delivery and feed variables D_{slo}, R_{slo} are assumed to be given.

The exact solution of MILP problem (16.37) is hardly possible in the realistic cases. Therefore we will consider how to apply BHA to this problem. We start that by considering batch scheduling optimization in a general MILP form.

16.4 APPLICATION OF BHA TO THE BATCH SCHEDULING PROBLEM, USING MILP AND GA

16.4.1 "String" Definition

We define a state m by fixing the vector

$$s(m) = (q(m), p(m)) \quad (16.42)$$

Here $q(m) = (q_{ij}(m), i = 1, \dots, I, j = 1, \dots, J)$ and $p(m) = (p_{ij}(m) i = 1, \dots, I, j = 1, \dots, J)$,

where

$$q_{ij}(m) = \begin{cases} 0, & \text{if operation } (i, j) \text{ is irrelevant} \\ 1, & \text{if operation } (i, j) \text{ is necessary} \\ q_{ij} \in \{0, 1\}, & \text{otherwise,} \end{cases} \quad (16.43)$$

and $p_{ij}(m)$ is an order-number of operation (i, j) . Here the symbol $\{0, 1\}$ denotes a Boolean variable.

We eliminate "zero q_{ij} " operations that are irrelevant in the sense of (16.43). As usual, in real-life batch scheduling problems, we do not know in advance which operations are necessary and which are irrelevant. Therefore we have to regard $q_{ij}(m)$ as an optimization variable.

The vector $q(m)$ is called an operation-vector, and the vector $p(m)$ an order-vector. Thus the operation-vector $q(m)$ defines which operations to perform and the order-vector $p(m)$ defines an order of operations.

For example, the operation-vector

$$q(m) = (q_{1,2} = 1, q_{3,4} = 1, q_{5,6} = 0, \dots) \quad (16.44)$$

means that at the m -th state operations $(1, 2)$, $(2, 3)$ are performed, and operation $(4, 5)$ is not performed.

The order-vector

$$p(m) = (p_{1,2} = 1, p_{3,4} = 2, \dots) \quad (16.45)$$

means that operation $(1, 2)$ is performed before operation $(3, 4)$, and so on.

In MILP terms the order vector $s(m)$ is defined by fixing Boolean variables W_{ijo}^S . If we start the operation (i, j) at the moment t_o then $W_{ijo}^S = 1$, otherwise, $W_{ijo}^S = 0$. The delivery/feed indicators a_{ilo}^D , a_{ilo}^R , time t_o^D , t_o^R , and parameters d_{io} , r_{io} are regarded as input data.

Constraints (16.21) can be coped with by introducing some additional variables

$$b_{jo1} = 1 - N_{jo} \geq 0 \quad (16.46)$$

$$b_{jo0} = N_{jo} \geq 0. \quad (16.47)$$

16.4.2 Reproductive Plan

If the population size is not limited to just a few "members", then the extended polynomial randomization with the "Delta" term (see subsection 11.5.4) seems reasonable because it includes

- "pure" heuristics (choose the fittest);
- Monte-Carlo (choose at random);
- linear randomization (fitness proportional choice).

More sophisticated randomization procedures including polynomials of higher degrees might be included as well, if preferable.

If we limit population size to just two members (the old and the new one), then we may also consider an exponential randomization of simulated annealing type, too (see 11.22).

16.4.3 Cross-Over and Mutation Operations

We may apply the same cross-over and mutation operations as described in Chapter 13.2.2. Two selected strings are split at some random position i_s . Then two new strings are created, by exchanging all the characters up to the split point i_s .

A mutation is said to be of n -th order if $n = 1, \dots, I$, elements are changed at random during one mutation operation. A mutation is feasible, if it meets all the constraints.

There are some specifics, too. It would not be natural to cross-over segments between the operation-vector $q(m)$ and order-vector $p(m)$. Therefore we cross-over segments of those vectors separately. If the operation-vector $q(m)$ and/or order-vector $p(m)$ contain some segments defining specific "traits" of the batch scheduling problem, then the cross-over operation will contribute a lot to the general optimization. Otherwise, it could be of some help "escaping" from the "local" minima. We may get the answer by extensive experimentation.

16.4.4 Getting Feasible Solutions

As usual, we get feasible solutions for fixed string $s(m)$ using the standard MILP software. One may also use specific round-off algorithms, such as this one:

- fix the string $s(m)$ and "round-off" to 1 the greatest component:

$$\max_o W_{ijo}^S = 1, \quad (16.48)$$

It follows from conditions (16.1)-(16.30) that in this case we get integer values of the variables N_{jo} , W_{ijo}^F , too.

The rounding-off operation (16.48) is simpler as compared with the general MILP procedures getting the first feasible solution.

16.5 AGGREGATE OF BATCH SCHEDULING PROBLEMS

We need an exponential time $T = C \cdot 2^n$ to solve a NP-complete n -dimensional problem. We need just $T = NC \cdot 2^{n/N}$ of time to solve N "parts" of this problem independently. The trouble is that the set of "partially" optimal solutions will not represent the general optimal solution of the whole problem. The objective of Aggregation is to "coordinate" the partial and the general solutions, to show how to get the general solutions uniting the partial ones.

By an aggregation we call splitting of an whole problem A into M slave-problems (aggregates) A^m , $m = 1, \dots, M$. The aggregates communicate via the variables of the master-problem.

By a Batch-Aggregate A^m we denote a union of some batch operations (i, j) , preferably close in space and time. The Batch-Master C defines communications between pairs (m, p) of Batch-Aggregates A^m and A^p , and between an aggregate A^m and the environment.

Mutual communications of the pair (m, p) are defined as a vector

$$C^{mp} = (D_{slo}^{mp}, R_{slo}^{mp}, t_o^{mpD}, t_o^{mpR}, o \in O^{mp}). \quad (16.49)$$

Here

D_{slo}^{mp} is what m delivers to p ,

R_{slo}^{mp} is what m receives from p ,

t_o^{mpD} is the delivery time,

t_o^{mpR} is the receive time,

O^{mp} is a set of these times.

The external communications of the aggregate m are defined as a vector

$$C^m = (D_{slo}^m, R_{slo}^m, t_o^{mD}, t_o^{mR}, o \in O^m). \quad (16.50)$$

Here

D_{slo}^m is what m delivers outside

R_{slo}^m is what m receives from outside

t_o^{mD} is the delivery time

t_o^{mR} is the receive time

O^m is a set of these times.

Complete communications of the aggregate m are defined as a vector

$$C(m) = (C^m, C^{mp}, p \in P^m). \quad (16.51)$$

Here P^m is a set of all "neighbors" of aggregate m .

We indicate "internal" parameters of aggregate m by an additional upper index m in expressions (16.1)-(16.30).

16.5.1 Slave-problem

In the m -th slave-problem, we maximize profit L^m of the aggregate m given the communication-vector $C(m)$. We denote the maximal profit as $L(m) = L(C(m))$. We may define $L(m)$ approximately by BHA-MILP algorithms (see Chapter 13).

16.5.2 Master-problem

Solving the master-problem we search for a communication-vector $C(m)$ which maximizes the total profit of all the aggregates

$$L^* = \max_{C(m), m=1, \dots, M} \sum_{m=1}^M L(C(m)) \quad (16.52)$$

satisfying the conditions

$$D_{slo}^{mp} = -R_{slo}^{pm} \quad (16.53)$$

$$t_o^{mpD} = t_o^{mpR} \quad (16.54)$$

$$p \in P^m, m = 1, \dots, M, o \in O^{mp}.$$

Master problem (16.52) is a continuous one. Therefore we may use Bayesian methods of global optimization directly, if the number of vector $C(m)$ components does not exceed, say, 20. Otherwise, BHA is applied.

16.5.3 Why Aggregate?

Denote by K^m the number of variables of the m -th aggregate. Then the number of variables of the whole problem is given by:

$$K = \sum_{m=1}^M K^m. \quad (16.55)$$

Fixing the communication-vector $C(m)$, we reduce the whole K -dimensional problem to a sequence of M K^m -dimensional problems.

Denote the number of components of the communication-vector $C(m)$ by K_C^m . Then the dimension of the master-problem is

$$K_C = \sum_{m=1}^M K_C^m. \quad (16.56)$$

Now we see that aggregation is useless, if

$$K_C \geq K. \quad (16.57)$$

The aggregation could be of some use, if K is much greater than K_C

$$K_C \ll K. \quad (16.58)$$

It follows from (16.58) that the aggregation may be efficient, if the communications are weak, meaning a small number of components of communication-vector. If this number is large, it means that we get a "non-aggregated" problem. If this number is small, it means that the aggregation might help. We say "might" because it is difficult to compare directly the complexity of discrete slave-problems and the complexity of a continuous master-problem.

16.5.4 Simplification of Master-problem

If we may sell or buy all the materials at any moment at fixed market prices, then we may maximize profits L^m of aggregates adding the costs of delivered materials and subtracting the costs of received ones. However, we have to define the optimal delivery and receive time t_o^{mpD} , t_o^{mpR} , anyway.

If the master-problem is optimized by Bayesian methods, one may use roughly approximate solutions of slave-problems, because these methods are relatively insensitive to "noise". We increase the accuracy towards the end of optimization, if needed.

16.5.5 Aggregate Complexity

The whole and the slave problems are assumed to be NP-complete¹ (see [79]) We assume that the master-problem is polynomial-time computable (see [79]). Then the optimization of the master-problem belongs to a family of NP-complete problems. We say that a real function $f(x)$ is polynomial-time computable, if for any given real number x and any precision n one may get in a polynomial time a rational number a_n such that

$$|a_n - f(x)| \leq 2^{-n}. \quad (16.59)$$

The optimization problem is said to be computable if for any precision n one may get a rational number a_n such that

$$|f(a_n) - \min_x f(x)| \leq 2^{-n}. \quad (16.60)$$

¹A problem belongs to a family P if there exists a polynomial-time algorithm of solution (see Subsection 2.2.5). We denote by $NP = P$ an assumption that all the problems belonging to the NP-complete family may be solved by polynomial-time algorithms. It follows from the definition, that no NP-complete problem can be solved by the polynomial-time algorithm, unless $NP = P$.

Solution of the master-problem in the sense of expression (16.60) is as difficult as an NP-complete problem even if the m -th derivative of f exists and is continuous on $[0,1]$ for all m (see [79]). Note that the precision n plays a similar role defining the complexity of continuous optimization as a number of variables k in discrete optimization (see Sub-section 2.2.5).

16.5.6 Aggregate Complexity and Knapsack Example

Whole Problem

We may illustrate the aggregate complexity better considering the simple knapsack problem.

$$\begin{aligned} \max_x \sum_{i=1}^n c_i x_i & \quad (16.61) \\ \sum_{i=1}^n g_i x_i & \leq g \\ x_i & \in \{0, 1\}. \end{aligned}$$

Here the objective depends on n Boolean variables x_i .

Aggregate-Problem

Denote $x^j = (x_1^j, \dots, x_{n_j}^j)$, $j = 1, \dots, N$,
where $x_1^1 = x_1, x_{n_1}^1 = x_{n_1}, x_1^2 = x_{n_1+1}, \dots, x_{n_2}^2 = x_{n_1+n_2}, x_3 = x_{n_1+n_2+1}, \dots$

Denote $g^j = (g_1^j, \dots, g_{n_j}^j)$, $j = 1, \dots, N$,
where $g_1^1 = g_1, g_{n_1}^1 = g_{n_1}, g_1^2 = g_{n_1+1}, \dots, g_{n_2}^2 = g_{n_1+n_2}, g_3 = g_{n_1+n_2+1}, \dots$

Denote y_j , $j = 1, \dots, N$ positive numbers such that $\sum_{j=1}^N y_j = g$. In this case, $x = (x^j, j = 1, \dots, N)$ and we may write

$$\max_x \sum_{j=1}^N \sum_{i=1}^{n_j} c_i^j x_i^j \quad (16.62)$$

$$\sum_{j=1}^N \sum_{i=1}^{n_j} g_i^j \leq y_j.$$

Slave-Problems

Whole knapsack problem (16.61) is split into N slave-problems $j = 1, \dots, N$

$$\begin{aligned} c_j(y_j) = \max_{x_j} \sum_{i=1}^{n_j} c_i^j x_i^j & \quad (16.63) \\ \sum_{i=1}^{n_j} g_i^j \leq y_j & \\ x_i^j \in \{0, 1\}, & \end{aligned}$$

Master-Problem

The slave-problems are controlled by the following master-problem:

$$\begin{aligned} \max_y \sum_{j=1}^N c_j(y_j) & \quad (16.64) \\ \sum_{j=1}^N y_j \leq g. & \end{aligned}$$

Aggregate Optimization

We need $K_j = 2^{n_j}$ iterations in optimizing the j -th slave-problem by "exhaustive" search. We need $K = \sum_{j=1}^N K_j$ iterations to optimize all the slave-problems.

The objective function of master-problem (16.64) is a step-function of N real variables y_j , $j = 1, \dots, N$. In optimizing the master-problem by exhaustive search, as many iterations are necessary as there are "steps". The number of steps depends not only on N but also on n_j , $j = 1, \dots, N$.

This implies that aggregation may be of no use if we apply an exhaustive search in optimizing the master-problem. Aggregation may help if we use Bayesian

methods to optimize this problem. The reason is that Bayesian methods are based on statistical models (for example, a Wiener model) which "smooth" the small "steps" of the master-problem objective function during the process of global optimization.

PART VI

**SOFTWARE FOR GLOBAL
OPTIMIZATION**

INTRODUCTION TO GLOBAL OPTIMIZATION SOFTWARE (GM)

17.1 BACKGROUND

The global optimization software was developed considering the results of international "competition" of different algorithms of global optimization (see [30]). Some experience in real life optimization problems was also used selecting the set of optimization algorithms. The set of algorithms of global optimization includes

- four versions of the Bayesian search,
- a version of clustering, a version of uniform deterministic grid,
- a version of pure Monte Carlo search.

Usually it is reasonable to start optimization by a global method and to finish it by some local method. An exception is two global algorithms: the Torn version of clustering [148], and the Zilinskas version of the Bayesian technique [148]. Both these algorithms contain some simple algorithms of local search. This means that the local search is not necessary for those two methods, but it may be useful.

There are three local optimization methods:

- a method of variable metrics type with Lagrangian multipliers and penalty functions for constrained optimization of smooth functions (see [137]),

- a method of simplex type of Nelder and Mead with penalty functions for constrained optimization of non-differentiable functions.
- a method of stochastic approximation type for "noisy" functions (see [100]).

Each subroutine represents a global or a local method. The choice of method has to follow the idea that the computational complexity of the method should roughly correspond to that of the objective function:

- For computationally "expensive" functions the Bayesian methods could be recommended. Those methods need a large amount of auxiliary calculations to make each observation more efficient.
- For "cheap" functions the simple grid methods, like Monte Carlo or a uniform deterministic grid (see [141]), can be better. Here observations are not so efficient, but auxiliary calculations are negligible. This explains a relative efficiency of simple methods when optimizing simple functions.
- The clustering techniques (see [148]) may be the best choice, if we expect the number of local minima to be small.
- A relatively simple Bayesian technique is available [148] for global optimization of one-dimensional functions;
- There are optimization problems where objective functions can be roughly represented as a sum of components depending on different variables. Here the Bayesian method of line search along the coordinates usually shows very good results. This method globally optimizes one variable at a time by one-dimensional Bayesian search. The difference of this method from other methods of global optimization is that it depends on the starting point. Thus a deviation from the global minimum can be made as small as desired by applying a multi-start procedure with different uniformly distributed starting points.

All the global methods optimize in a rectangular region. Therefore we represent the linear and non-linear inequality constraints as some penalty functions. The same applies to the local method of stochastic approximation type. In local methods of simplex and variable metrics type the linear and the non-linear constraints can be defined directly. This may be done by constraint subroutines, supplied by the user in addition to the objective function.

The global optimization software is in four versions:

- portable Fortran Library,
- interactive software for Microsoft C&Fortran compiler and DOS operating system,
- interactive software for Turbo C compiler and DOS operating system,
- interactive software for C++ compiler and UNIX operating system and X-Window system.

The library of portable Fortran subroutines (LINUX 1.2.8. and DOS versions) and the interactive UNIX C++ software (LINUX 1.2.8. version), are in the enclosed disk.

17.2 TYPICAL EXAMPLES

Many examples of applications are related to the optimization of parameters of mathematical models, represented as some systems of non-linear differential equations. The objective function $f(x)$ here depends on a solution of equations. Variables x represent the controlled parameters of the system. A good illustration of this family of problems is the following three examples :

- Maximization of the general yield of differential amplifiers [100] (see Section 5.2).
- Optimization of the mechanical system of shock-absorber [100] (see Section 5.3).
- Estimation of the parameters of non-linear regression of an immunological model [100] (see Section 5.4).

The last example suggests a broad area for applications of global optimization. It is well known that in non-linear regression a square deviation as well as a likelihood function could be multi-modal for some data.

The number of local minima may be very large, even in simple cases. The examples are estimation of unknown parameters of ARFIMA (see Chapter 6) and ANN (see Chapter 10) models.

There are other important families of global optimization problems, namely:

- A large source of difficult global optimization problems is engineering design. Here we are optimizing parameters of some mathematical models, usually non-linear. Optimization of composite laminates [104] (see Section 5.6) serves as an example.
- Many laws of nature could be defined in terms of global optimization. An example is the "Disk" problem: minimization of potential energy of organic molecules [104] (see Section 5.7).
- We are often not able to describe the behavior of new materials and technologies by mathematical models, because the corresponding information and knowledge is not available. Here one can optimize by direct experiments, changing the control variables and observing the results. An example is the planning of extremal experiments of thermostable polymeric composition [104] (see Section 5.8).

A number of examples of GM software applications are given in [100, 94].

Bayesian algorithms of the global optimization software GM are an essential part of BHA, needed to optimize the parameters of randomized heuristics (see Chapters 12, 14, 15, and 16).

17.3 DIFFERENT VERSIONS

17.3.1 Portable Fortran Library

We start a description of global optimization software from the Fortran library (see Chapter 18), because it is easy, and in the Turbo C and UNIX C++ versions mostly the same algorithms are used and their parameters have a similar meaning. The UNIX C++ version is described in Chapter 19. However, a UNIX user may obtain some useful information by reading the Fortran library description. The portable Fortran version can run on any computer with a standard Fortran compiler. Users should represent objective functions in the form of `FUNCTION FI(X,N)`, where `X` is an array of variables, `N` is its dimension.

Rectangular constraints are given by the lower bound array A and the upper bound array B. Using local methods of simplex and variable metrics type one may represent constraints by subroutine CONSTR(X,N,G,MC), too. Here, G is an array of length MC which contains the values of constraints at the point X, and MC is the number of constraints.

17.3.2 Interactive DOS C&Fortran version

Here the objective function can be written in one of the two forms: Fortran or C. This system is for users which prefer to represent objective functions, both in C and in Fortran. This version needs Microsoft C&Fortran compilers [96].

In this version, there is a possibility of vector optimization applying the idea of Pareto optimality. The Pareto set is approximately defined by comparing all the "Bayesian" or "Uniform" observations and excluding the dominated ones.

17.3.3 Interactive DOS Turbo C version

This version is for objective functions represented in C. It needs a Turbo C compiler. A user represents the objective function $f(x)$ as some C subroutine *f.i.c.*

In both C & Fortran, and Turbo C interactive systems users can select a global or a local method by a menu system. The current results can be observed in both, tabular and graphical, forms. There are two graphical forms:

- GRAPH: shows how the best value of the objective function depends on the observation number.
- PROJECTION: shows how current objective function value depends on a fixed variable.

The main advantage of the first version, the Fortran library, is its portability. A disadvantage is that interactive possibilities are very limited. This means that this version can be easily used for some well defined optimization problems, but not for preliminary investigations, where the interaction is essential.

An advantage of the second (C&Fortran) and the third (Turbo C) version is rather good interactive facilities. A disadvantage is that both these versions

can be used only on PC. It is all right for a preliminary investigation and for solution of small scale problems. For real life global optimization problems the limitations of DOS operating system and computational power of a single PC can be too restricting.

17.3.4 Interactive UNIX C++ version

It is a global optimization software designed for UNIX and X Window systems. It has the same interactive facilities as a DOS Turbo C version, plus portability to almost any computer, including super computers and some parallel computers.

Concluding this chapter note that two global optimization software versions and application examples will be described in detail:

- the Fortran library, in the next chapter,
- the interactive UNIX C++ software, in Chapter 19,
- application examples of the UNIX C++ version in Chapter 20.

Most of the algorithms and their parameters described in the Fortran version remain in the UNIX one. Therefore a glance at the next chapter describing the Fortran library could be useful before reading about the structure and application examples of the interactive UNIX C++ version.

18

PORTABLE FORTRAN LIBRARY FOR CONTINUOUS GLOBAL OPTIMIZATION

18.1 INTRODUCTION

The purpose is to minimize a continuous function

$$f(x), \quad x = (x_1, \dots, x_n), \quad (18.1)$$

where $x \in A \subset R^n$.

It is assumed for most methods that the set A is a rectangular parallelepiped

$$A = \{z : a_i \leq x_i \leq b_i, \quad i = 2, \dots, n\}. \quad (18.2)$$

The more general cases are approximately reduced to expression (18.2) using the penalty function techniques (see [100]). A distinction of the software is implementation of the methods which can be regarded as optimal in the sense of average deviation. The next observation is defined by minimizing the risk function (expected deviation from a the global minimum). The number of observations is fixed, as usual.

The procedures that minimize the risk function are as simple as possible (see Section 4.4), but there exist some natural limits. The risk function cannot be reduced to the uni-modal one, if the convergence to a global minimum of any continuous function is to be provided. Consequently it seems that the global Bayesian method described in [100] is, possibly, the simplest one which minimizes the average deviation and converges to a minimum. If the objective function $f(x)$ is simple enough, then it is better not to minimize the risk function at all but to take more observations, that need not to be planned optimally.

In such a case, a good idea is to use a uniform search, for example, of LP type (see [141]). We lose the average optimality but still have the convergence. Even the LP-search can be too expensive, if the observations are very cheap. Then we use the uniform random (Monte Carlo) search which is very simple, but we obtain the convergence only in the probabilistic sense. Both the LP search (LPMIN) and the uniform random search (MIG1,MIG2) are in the library.

The method of clustering [148] is included as a good heuristic technique under the title GLOPT. It usually works well, if the number of local minima is small and known in advance, and if the "attraction area" of the global minimum is large. The clustering GLOPT is comparable to LP-search LPMIN by the computing time .

The global line search EXKOR is a "semi-global" method designed for approximately "additive" functions (see [148]). Note that EXKOR includes elements of visualization. Using EXKOR one may conveniently see how the objective depends on a variable. That is important for the preliminary investigation.

The global methods search a whole area, as usual. Otherwise one can miss the global minimum. Obviously, the global search is not a most efficient way for local optimization. Therefore, in the GM software the best result of global search is defined by default as an initial point for a local search completing the optimization process.

For local optimization of non-linear constrained continuously differentiable functions without noise the methods of variable metrics (VM) type seem to be the best. A version of VM described in [11] is included under the name REQP¹. A specific VM version for rectangular constraints is defined as MIVAR4 (see [147]).

For local optimization of non-differentiable functions with non-linear constraints the simplex type method is used [61] under the name FLEXI. If there are some reasons to expect that the influence of some variables and their groups is considerably greater than that of others, then the method LPMIN should be used for analysis of the structure before we start the regular optimization. The LPMIN orders variables by their "importance" using a sort of variance analysis techniques (see [136]).

¹In the interactive C and C++ versions a different VM algorithm is implemented (see [137]) under the name NLP.

For local optimization of uni-modal functions with a noise is a stochastic approximation type algorithm with the "Bayesian" step length control under the name LBAYES (see [100]).

By adjusting standard machine-dependent constants, the software can be adapted to any computer with a standard FORTRAN compiler, since no machine-dependent routines are used.

18.2 PRELIMINARY DISCUSSION

18.2.1 Installation

DOS Fortran 77 version of GM

Installing:

- copy the file 'gmf.arj' by the DOS command

```
> copy a: gmf.arj
```
- extract the GM files from the 'gmf.arj' archive

```
> arj e gmf.arj
```

This version was tested using the Digital Research Fortran-77 Version 4.1 and LINK-86 Linkage Editor Version 1.5f, Digital Research, Inc.

The example:

- compile

```
> f77 ex9.for
```

```
> f77 i1mach1.for
```

```
> f77 exkor.for
```
- link

```
> link86 ex9,i1mach1,exkor
```

- run
- > ex9

Here the file 'ex9.for' is an example of the Main program using EXKOR method (see Subsection 18.3.8) to minimize FURASN function (see expression 18.3). The file 'exkor.for' is the source of EXKOR. The file 'ilmach1.for' contains portability codes (see Subsection 18.3.13) and other service routines such as independent random number generator, various test functions, etc.

LINUX Fortran 77 version of GM

Installing:

- copy the file 'gmfl.tgz' by the LINUX command
- > mcopy a: gmfl.tgz
- extract the GM files from the 'gmfl.tgz' archive
- > tar -zxf gmfl.tgz

This version was tested using standard LINUX f77-style shell script 'f77' to compile and load Fortran, C, and assembly codes.

The example:

- compile and load
- > f77 ex9.f ilmach1.f exkor.f
- run
- > a.out

Note that here the file extensions are 'f'.

18.2.2 Parameters

The parameters X, A, B, N, FM, IPAR, PAR, IPA, IPAA are used in all sub-routines. Here

X is an array of length N which defines the coordinates of a point being considered (initial, optimal or current),
A is an array of length N which defines the lower bounds of X,
B is an array of length N which defines the upper bounds of X,
N is the number of variables (dimension of X) usually $N \leq 20$,
FM is the value of the function $FI(X, N)$ at the point X,
IPAR is the array of length 30 which defines the real control parameters,
IPA is a shift of integer control parameters,
IPAA is a shift of real control parameters.

If only one method is used, then both shifts are zero: $IPA = 0$ and $IPAA = 0$.

If several methods are used sequentially, then a shift for the next method must be equal to the sum of numbers of control parameters used before by other methods. The number of control parameters of different methods is given in Table 18.1. For all the methods the first integer control parameter is the printing parameter:

$IPAR(IPA + 1) = IPR$
($IPA = 0$ if only one method is used).

If $IPR < 0$,
then only diagnostic messages are printed.
If $IPR = 0$,
then the initial data, the final results and diagnostic messages are printed.
If $IPR > 0$,
then not only those but also the results of each IPR-th iteration are printed.

The meaning of other control parameters will be explained when describing the corresponding subroutines.

18.2.3 List of Methods

Table 18.1 Table of Parameters

No	Name	Method	<u>No. of control parameters</u>	
			Integer IPAR	Real PAR
<u>Global optimization with rectangular constants</u>				
1	BAYES1	Bayesian (see [100])	3	0
2	UNT	Extrapolation (see [166])	4	0
3	LPMIN	Uniform deterministic (see [136, 141])	N+3	0
4	GLOPT	Clustering (see [148])	3	0
5	MIG1, MIG2	Uniform random	2	0
6	EXTR	Bayesian one-dimensional (see[165])	3	2
7	EXKOR	Extension of EXTR to multi-dimensional case (see[165])	5	N+1
<u>Local optimization</u>				
8	MIVAR4	Variable metric with rectangular constants (see [147])	4	4
9	REQP	Variable metrics with non-linear constraints (see [11])	4	4
10	FLEXI	simplex with non-linear constraints (see [61])	4	2
<u>Local optimization with noise</u>				
11	LBAYES	stochastic approximation with Bayes step length and rectangular constraints (see [100])	3	2

We see from Table 18.1 that to set the algorithm parameters one must make many adjustments. The setting of parameters can be made easy using the "ready-made" examples including sequences of different algorithms. We may chose the sub-sequence we wish just by "commenting-out" the redundant algorithms. For example, the file 'gmfl/ex8.f' in the enclosed disk includes seven

algorithms named as BAYES1, MIVAR4, LBAYES, FLEXI, REQF, UNT, GLOPT. The file 'gmfl/ex9.f' includes only one algorithm EXKOR.

18.2.4 Common Blocks

In the array /BS1/Y(100) there are values of the function $FI(X, N)$, where X is defined by the array XN of length $MN=N*M$.

In the array /STATIS/IFAIL, IT, IM, M:

IFAIL is a control indicator:

- if the initial data is not correct, then $IFAIL = 10$ and return to the main program,
- if the initial data is correct, then $IFAIL \neq 10$ and shows the number of the termination criterion,

IT is the number of iterations,

IM is the number of the optional iteration (where the best point was found),

M is the number of function evaluations (observations).

18.2.5 Objective Function

The function to be minimized should be represented as a real function $FI(X,N)$. In most methods, only the lower and upper bounds should be fixed by arrays A and B .

In methods with non-linear constraints the subroutine

CONSTR (X, N, G, MC)

should be used, where

G is a one-dimensional array of length MC which defines the constraints at the point X

MC is the number of constraints.

It is well known that the local methods of optimization in general are sensitive to the scales of variables. The parameters of local methods in this package are usually adjusted to the case when $A(I) = -1$ and $B(I) = 1$, $I = 1, N$. Therefore, it can be useful to reduce the rectangular constraints $A \leq X \leq B$

to the N -dimensional hypercube $[-1, 1]^N$, which can be arranged using the reduction formulae²).

The reduction formulae are as follows

$$X(I) = \frac{2}{B(I) - A(I)} X0(I) - \frac{B(I) + A(I)}{B(I) - A(I)}, \quad I = 1, N,$$

where $X0$ are original variables, and X are variables scaled to fit into $[-1, 1]^N$.

EXAMPLE. In most of the following examples, the following test function $f(x)$ ³ is used

$$f(x) = 2/N \sum_{i=1}^N (x_i^2 - \cos(18x_i)) \quad (18.3)$$

with $N = 2$, $x_1 \in [-0.25; 0.5]$, $x_2 \in [-0.125; 0.625]$,

which is represented by the subroutine function FURASN:

```

FUNCTION FURASN(X,N)
DIMENSION X(N)
NN=N
F=0.
DO 10 I=1,NN
XI=X(I)
10 F=F+XI*XI-COS(18.*XI)
AN=NN
FURASN=F*(2./AN)
RETURN
END

```

Figure 18.1 Objective Function FURASN

²In the case of LBAYES those formulae are included in the algorithm, therefore no additional reduction is needed.

³This is the well known Rastrigin [127] test function.

18.2.6 Main Program

The main program defines the input data and the sequence of methods. At the beginning, we should choose, from Table 18.1 (or using the ready-made example 'gmfl/ex8.f'), a desirable sequence of methods of optimization and analysis. Then a function $FI(X, N)$ should be prepared that evaluates the objective function at a fixed point X .

If necessary, the subroutine $CONSTR(X, N, G, MC)$ is included which evaluates the constraints at the point X . The length of arrays usually depends on the subroutines. The exception is the arrays $IPAR$ and PAR . The length of these arrays is always 30. The parameters of methods are included in the arrays $IPAR$ and PAR in accordance with the given sequence of methods. The formal parameters IPA and $IPAA$ are fixed using the rules given in the previous section. In the case when only one method is used, $IPA = IPAA = 0$.

18.2.7 Example of the Main Program

The following test problem is considered. The global minimum of test function (18.3) should be determined using the global Bayesian method $BAYES1$. Then the results of global search should be corrected by the local method of variable metrics $MIVAR4$.

The test function is represented as the function $FURASN(X, N)$.
The arrays are: $X, A, B, XN, HES, IPAR, PAR$

It follows from Table 18.1 that in the subroutine $BAYES1$ three integer parameters, and in the subroutine $MIVAR4$, four integer parameters are used. In $MIVAR4$ four real parameters should be defined, too. This means that seven elements of $IPAR$ and four elements of PAR should be fixed.

The main program is in Figure 18.2

18.3 DESCRIPTION OF ROUTINES

In this chapter a general description of routines corresponding to different methods of global and local optimization will be given. The general description of routines includes the title and purpose of a routine, restrictions and accuracy. It also shows how to use the routine. Simple examples are provided.

```

program main
dimension x(2),a(2),b(2),xn(200),hes(3),ipar(30),par(30)
data n,nm,nh,ipa,ipaa/2,200,3,0,0/
data a/-0.25,-0.125/,b/0.5,0.625/
data ipar/0,100,5,0,100,2,100,23*0/
call bayes1(x,a,b,n,xn,nm,fm,ipar,ipa)
ipa=3
call miavar4(x,a,b,n,hes,nh,fm,ipar,par,ipa,ipaa)
stop
end
function fi(x,n)
dimension x(n)
fi=furasn(x,n)
return
end

```

Figure 18.2 Example of MAIN

The FORTRAN codes are given separately in the diskette, because the complete listings are needed only for advanced users who want to check or to change the parameters and procedures of optimization. In the general description of algorithms and programs only the most important information is provided.

18.3.1 BAYES1: the global Bayesian method by Mockus

PURPOSE. To find the global minimum of a continuous function (18.1) of N variables defined on a rectangular parallelepiped (18.2).

RESTRICTIONS. In terms of efficiency this program becomes increasingly less successful as the dimensions of the parallelepiped are increasing. Consequently the dimensions must be as small as possible.

The global Bayesian method uses a considerable amount of auxiliary calculation. As a result no more than 1000 function evaluations can be performed. This means that BAYES1 can be efficiently used only when the function $f(x)$ is

difficult (takes more CPU time than the calculation of the coordinates of the next point). The global method search the whole area. Therefore, as usual, it finds a point in the neighborhood of a global minimum but not the exact minimum. Therefore some local methods are used to carry out the local minimization.

ACCURACY. The global method provides the minimal average deviation in accordance with a given statistical model (see [100]) and the convergence to the global minimum for any continuous function. This implies that if we solve many problems, the average error will be as small as possible. However, for some fixed samples it can be large, if the iteration number is limited.

HOW TO USE THE METHOD

CALL BAYES1 (X, A, B, N, XN, NM, FM, IPAR, IPA)

where the input is: A, B, N, NM, IPAR, IPA

and the output is X, XN, FM.

XN is an array of length $NM = N * M$ which contains coordinates of M points of function evaluation.

In the main program the following arrays should be described:

A(N), B(N), X(N), XN(NM), IPAR(30)

$N \leq 20$

IPAR(1) = IPR is a printing parameter

IPAR(2) = M is the number of function evaluation $M \leq 1000$.

IPAR(3) = LT is the number of initial points which are uniformly distributed by LP sequences of Sobolj (1969) $0 < LT \leq m$.

EXAMPLE. The program locates the minimum of the multi-modal function (9.5.1). IPR = 0, M = 100, LT = 5. The initial information is fixed by DATA.

The main of BAYES1 is in Figure 18.3 and the output is illustrated by Figure 18.4.

```

program exbayes1
dimension x(2),a(2),b(2),xn(200),ipar(30)
data n,nm,ipa/2,200,0/,a/-0.25,-0.125/,b/0.5,0.625/
data ipar/0,100,5,27*0/
call bayes1(x,a,b,n,xn,nm,fm,ipar,ipa)
stop
end
function fi(x,n)
dimension x(n)
fi=furasn(x,n)
return
end

```

Figure 18.3 Example of BAYES1 main

18.3.2 UNT: The global method of extrapolation type by Zilinskas

PURPOSE. To find the global minimum of a continuous function (18.1) of N variables defined on the rectangular parallelepiped (18.2). **RESTRICTIONS.** The restrictions are the same as in the BAYES1 method except that if the function is differentiable, then the local search of variable metrics type can be directly incorporated in UNT. **ACCURACY.** The method provides the minimal average deviation in accordance with a set of assumptions given by [166]. All other accuracy considerations are similar to those in the BAYES1 method. **HOW TO USE THE METHOD**

CALL UNT (X, A, B, N, XN, NM, FM, IPAR, IPA)

where the input is: A, B, N, NM, IPAR, IPA and the output is X, XN, FM.

XN is an array of length $NM = N * M$ which contains coordinates of M points of function evaluation.

In the main program the following arrays should be described:

A(N), B(N), X(N), XN(NM), IPAR(30)
 $N \leq 20$

```
          B A Y E S 1

I N I T I A L  D A T A

NUMBER OF VARIABLES           N =      2

PRINTING PARAMETER           IPR =      0
NUMBER OF FUNCTION EVALUATIONS M =    100
NUMBER OF INITIAL POINTS     LT =      5

VECTOR OF LOWER BOUNDS (A) FOR X
  -.25000000E+00  -.12500000E+00
VECTOR OF UPPER BOUNDS (B) FOR X
  .50000000E+00  .62500000E+00

R E S U L T S

OPTIMAL FUNCTION VALUE FM =  -.19903086E+01
OBTAINED IN NR =      81
OPTIMAL POINT
  .22979677E-02  -.73658228E-02

NUMBER OF FUNCTION EVALUATIONS L =    100

          BAYES1 TERMINATED
```

Figure 18.4 Example of BAYES1 output

IPAR(1) = IPR is a printing parameter

IPAR(2) = M is the maximal number of function evaluation,
 $M \leq 500$.

IPAR(3) = LT is the number of RANDOM uniformly distributed
initial points, $LT \geq 30$

IPAR(4) = ML is the maximal number of local minima,
 $0 < ML \leq 20$.

EXAMPLE. The program locates the minimum of the multi-modal function (9.5.1). $IPR = 0$, $M = 500$, $ML = 5$. The initial information is fixed by DATA.

The main of UNT is in Figure 18.5 and the output is illustrated by Figure 18.6.

```

program exunt
dimension x(2),a(2),b(2),xn(200),ipar(30)
data n,nm,ipa/2,200,0/,a/-0.25,-0.125/,b/0.5,0.625/
data ipar/0,100,30,5,26*0/
call unt(x,a,b,n,xn,nm,fm,ipar,ipa)
stop
end
function fi(x,n)
dimension x(n)
fi=furasn(x,n)
return
end

```

Figure 18.5 Example of UNT main

18.3.3 LPMIN: The global method of uniform search by Sobolj, Saltenis and Dzemyda

PURPOSE. To locate the global minimum of a continuous function (18.1) of N variables defined on the rectangular parallelepiped (18.2).

RESTRICTIONS. The restrictions are similar to those in the BAYES1 method with the exception that LPMIN is using less auxiliary calculations and therefore it can be recommended for minimizing simpler functions than BAYES1 or UNT.

ACCURACY. The method provides a convergence to the minimum. The average deviation is usually considerably greater than in methods BAYES1 and UNT.

HOW TO USE THE METHOD

```
      U N T

      I N I T I A L   D A T A

      NUMBER OF VARIABLES                N =      2

      PRINTING PARAMETER                 IPR =      0
      MAXIMUM NUMBER OF FUNCTION EVALUATIONS M =    100
      NUMBER OF INITIAL POINTS          LT =     30
      MAXIMUM NUMBER OF LOCAL MINIMA    ML =      5

      VECTOR OF LOWER BOUNDS (A) FOR X
      -.25000000E+00  -.12500000E+00
      VECTOR OF UPPER BOUNDS (B) FOR X
      .50000000E+00  .62500000E+00

      R E S U L T S

      OPTIMAL FUNCTION VALUE FM = -.19660562E+01
      OPTIMAL POINT
      -.13996658E-01  -.36565200E-02

      LOCAL OPTIMA
      FUNCTION VALUE      POINT
      -.19660562E+01  -.13996658E-01  -.36565200E-02

      NUMBER OF FUNCTION EVALUATIONS L =    100

      IFAIL = 2.  TERMINATION CRITERION: NUMBER
      OF FUNCTION EVALUATIONS EQUALS M

      UNT TERMINATED
```

Figure 18.6 Example of UNT output

CALL LPMIN (X, A, B, N, NM, FM, IPAR, IPA)

where the input is: A, B, N, NM, IPAR, IPA and the output is X, XN, FM.

XN is an array of length $NM = N * M$ which contains coordinates of M points of function evaluation.

In the main program the following arrays should be described:

A(N), B(N), X(N), XN(NM), IPAR(30)

$N \leq 20$

IPAR(1)=IPAR is a printing parameter

IPAR(2)=M is the indicator of structure analysis. IF $M \neq 0$, then no analysis of structure is performed. If $10 \leq M \leq 300$, then the results of M observations are used to number the variables in decreasing importance order. If $M = 0$, then the order of variables should be fixed by the user in accordance with his opinion on the importance.

IPAR(3)= ML is number of steps of LP search

IPAR(4)	}	are the number of variables in order of decreasing importance which should be fixed when IPAR(2)=M=0.
.....		
IPAR(N+3)		

EXAMPLE. The program locates the global minimum and makes the structure analysis of multi-modal function (18.3).

The main of LPMIN is in Figure 18.7 and the output is illustrated by Figure 18.8.

18.3.4 GLOPT: The global method of clustering type by Törn

PURPOSE. To find the global minimum of a continuous function (9.1.) of N variables defined on the rectangular parallelepiped (18.2).

RESTRICTIONS. The restrictions are the same as in the BAYES1, UNT, LPMIN with the exception that GLOPT uses a smaller number of auxiliary calculations. Therefore it can be recommended for minimizing simpler functions as compared with all the previous methods, if the convergence is not important

```

program exlpmin
dimension x(2),a(2),b(2),xn(100),ipar(30)
data n,nm,ipa/2,100,0/,a/-0.25,-0.125/,b/0.5,0.625/
data ipar/0,50,1000,27*0/
call lpmin(x,a,b,n,xn,nm,fm,ipar,ipa)
stop
end
function fi(x,n)
dimension x(n)
fi=furasn(x,n)
return
end

```

Figure 18.7 Example of LPMIN main

ACCURACY. The convergence to the global minimum is not provided but the accuracy usually satisfies practical needs if the number of local minima is not large.

HOW TO USE THE METHOD

CALL GLOPT (X, A, B, N, FM, IPAR, IPA)

where the input is: A, B, N,IPAR, IPA and the output is X, FM.

In the main program the following arrays should be described:

A(N), B(N), X(N), IPAR(30)

$N \leq 20$

IPAR(1) = IPR is a printing parameter

IPAR(2) = M is the maximal number of function evaluation,

$M \leq 10000$.

IPAR(3) = LT is the number of random uniformly distributed initial points,

$LT \leq 150$, recommended LT is about double the number of an expected number of local minima.

EXAMPLE. The program locates the minimum of the multi-modal function (18.3).

```

      L P M I N

I N I T I A L D A T A

NUMBER OF VARIABLES                N =      2
PRINTING PARAMETER                 IPR =      0
NUMBER OF FUNCTION EVALUATIONS FOR ANALYSIS    M =    50
NUMBER OF FUNCTION EVALUATIONS FOR LP-SEARCH  ML =   1000
VECTOR OF LOWER BOUNDS (A) FOR X
  -.25000000E+00  -.12500000E+00
VECTOR OF UPPER BOUNDS (B) FOR X
  .50000000E+00  .62500000E+00

LP-SEARCH WITH ANALYSIS

RESULTS OF ANALYSIS

VARIABLES BY DECREASING INFLUENCE    2  1
OPTIMAL FUNCTION VALUE FM =  -.19209301E+01
OBTAINED IN NR =    10
OPTIMAL POINT   -.15625000E-01   .15625000E-01

RESULTS OF LP-SEARCH

OPTIMAL FUNCTION VALUE FM =  -.19915358E+01
OBTAINED IN NR =    404
OPTIMAL POINT   .63476562E-02   -.34179688E-02

R E S U L T S

OPTIMAL FUNCTION VALUE FM =  -.19915358E+01
OPTIMAL POINT   .63476562E-02   -.34179688E-02
NUMBER OF FUNCTION EVALUATIONS L =  1050

      L P M I N T E R M I N A T E D

```

Figure 18.8 Example of LPMIN output

The main of GLOPT is in Figure 18.9 and the output is illustrated by Figure 18.10.

```
program exglopt
dimension x(2),a(2),b(2),xn(1000),ipar(30)
data n,ipa/2,0/,a/-0.25,-0.125/,b/0.5,0.625/
data ipar/0,1000,10,27*0/
call glopt(x,a,b,n,fm,ipar,ipa)
stop
end
function fi(x,n)
dimension x(n)
fi=furasn(x,n)
return
end
```

Figure 18.9 Example of GLOPT main

18.3.5 MIG1: The global method of Monte Carlo (uniform random search)

PURPOSE. To find the global minimum of a continuous function (18.1) of N variables defined on the rectangular parallelepiped (18.2).

RESTRICTIONS. Can be recommended to minimize very simple functions, say, less than 1 sec of CPU times.

ACCURACY. The method converges in probability to the global minimum of continuous functions. The average deviation is considerably greater compared with global methods using the same number of function evaluations.

HOW TO USE THE METHOD

CALL MIGI (X, A, B, N, FM, IPAR, IPA)

where the input is: A, B, N, IPAR, IPA and the output is X, FM.

```

      G L O P T

I N I T I A L   D A T A

NUMBER OF VARIABLES              N =      2

PRINTING PARAMETER              IPR =      0
MAXIMUM NUMBER OF FUNCTION EVALUATIONS  M = 1000
NUMBER OF INITIAL POINTS        LT =     10

VECTOR OF LOWER BOUNDS (A) FOR X
  -.25000000E+00  -.12500000E+00
VECTOR OF UPPER BOUNDS (B) FOR X
  .50000000E+00  .62500000E+00

R E S U L T S

OPTIMAL FUNCTION VALUE FM =  -.19999995E+01
OPTIMAL POINT
  -.14713220E-04  -.48472517E-04

NUMBER OF FUNCTION EVALUATIONS L =   558

IFAIL = 0.  TERMINATION CRITERION: NUMBER OF
ITERATIONS EQUALS 20

      GLOPT TERMINATED

```

Figure 18.10 Example of GLOPT output

In the main program the following arrays should be described:

A(N), B(N), X(N), IPAR(30), X(N)

$N \leq 100$

IPAR(1) = IPR is a printing parameter

IPAR(2) = M is the number of function evaluation.

EXAMPLE. The program locates the minimum of the multi-modal function (18.3).

The main of MIG is in Figure 18.11 and the output is illustrated by Figure 18.12.

```
program exmig
dimension x(2),a(2),b(2),xn(1000),ipar(30)
data n,ipa/2,0/,a/-0.25,-0.125/,b/0.5,0.625/
data ipar/0,1000,28*0/
call mig1(x,a,b,n,fm,ipar,ipa)
stop
end
function fi(x,n)
dimension x(n)
fi=furasn(x,n)
return
end
```

Figure 18.11 Example of MIG main

18.3.6 MIG2: A modified version of MIG1

This is exactly the same method as MIG1 except that the coordinates of the points of all M function evaluations are stored in the array XN of length $NM=N*M$. The corresponding values of functions are stored into the array /BS1/Y(1000).

HOW TO USE THE METHOD

CALL MIG2 (X, A, B, N, XN, NM, FM, IPAR, IPA)

where the input is: A, B, N, NM, IPAR, IPA and the output is X, XN, FM.

In the main program the following arrays should be described:

A(N), B(N), XN(NM), IPAR(30), X(N)
 $N \leq 20$

```

      M I G 1

  I N I T I A L  D A T A

  NUMBER OF VARIABLES           N =      2

  PRINTING PARAMETER           IPR =      0
  NUMBER OF FUNCTION EVALUATIONS M = 1000

  VECTOR OF LOWER BOUNDS (A) FOR X
    -.25000000E+00  -.12500000E+00
  VECTOR OF UPPER BOUNDS (B) FOR X
    .50000000E+00  .62500000E+00

  R E S U L T S

  OPTIMAL FUNCTION VALUE FM =
  -.19686284E+01 OBTAINED IN NR =  411
  OPTIMAL POINT
    .76193810E-02  .11618406E-01

  NUMBER OF FUNCTION EVALUATIONS L = 1000

      MIG1 TERMINATED

```

Figure 18.12 Example of MIG output

IPAR(2)=M is the number of function evaluations, $M \leq 1000$.

18.3.7 EXTR: The global one-dimensional method by Zilinskas

PURPOSE. To locate the global minimum of a continuous function of one variable in a closed interval.

RESTRICTIONS. Restrictions are similar to those of multi-dimensional global methods. The local search is included into EXTR.

ACCURACY. The method provides the minimal average deviation from the minimum under the assumption that the objective function can be regarded as a sample of a Wiener process and converges to the minimum of continuous functions.

HOW TO USE THE METHOD

CALL EXTR (X, A, B, FM, IPAR, IPA, IPAA)

where the input is: A, B, N, IPAR, IPA and the input is: A, B, IPAR, PAR, IPA, IPAA
and the output is: FM, X

IPAR(1)=IPR is a printing parameter
IPAR(2)=M is the maximal number of function evaluation, $M \leq 500$.
IPAR(3)=LT is the number of initial points which are random uniformly distributed points,
 $LT \geq 6$, recommended $LT = 6$.
PAR(1)=EPS1 is the accuracy of minimization,
PAR(2)=EPS2 is the accuracy of the point to be minimized.

In the main program the following arrays should be described:
IPAR(30), PAR(30).

EXAMPLE. The program locates the minimum of the multi-modal function

$$f(x) = - \sum_{i=1}^r i \sin(i+1)x + i$$

with $x \in [-10, 10]$.

IPR=0, M=200, LT=6, EPS1= 10^6 , EPS2= 10^6 .

The main of EXTR is in Figure 18.13 and the output is illustrated by Figure 18.14.

```

program exextr
dimension ipar(30),par(30)
data ipa,ipaa/0,0/,a,b/-10.,10./
data ipar/0,200,6,27*0/,par/2*1.e-6,28*0./
call extr(x,a,b,fm,ipar,par,ipa,ipaa)
stop
end
function fi(x,n)
a=0
do 2 i=1,5
ai=float(i)
2 a=a-ai*sin((ai+1.)*x+ai)
fi=a
return
end

```

Figure 18.13 Example of EXTR main

18.3.8 EXKOR: The semi-global multi-dimensional method by Zilinskas

PURPOSE. To locate the global minimum of continuous function using EXTR along each coordinate one-by-one.

RESTRICTIONS. Restrictions are similar to those of multi-dimensional global methods. The local search is included into EXTR.

ACCURACY. The method approximates the global minimum of roughly additive functions such that $f(x) = \sum_{i=1}^m f_i(x_i) + c f_0(x)$ and $c > 0$ is small.

HOW TO USE THE METHOD

CALL EXKOR(X, A, B, N, FM, IPAR, IPA, IPAA)

```
      E X T R

      I N I T I A L   D A T A

      PRINTING PARAMETER   IPR =      0
      MAXIMUM NUMBER OF FUNCTION EVALUATIONS   M =    200
      NUMBER OF FUNCTION EVALUATIONS
      FOR PARAMETER ESTIMATION   LT =      6
      ACCURACY OF OPTIMAL FUNCTION VALUE   EPS1 = .10000000E-05
      ACCURACY OF OPTIMAL POINT           EPS2 = .10000000E-05
      LOWER BOUND (A) FOR X = -.10000000E+02
      UPPER BOUND (B) FOR X = .10000000E+02

      R E S U L T S

      OPTIMAL FUNCTION VALUE FM =  -.12031250E+02
      OPTIMAL POINT           XM =  .57918034E+01

      LOCAL OPTIMA
      POINT           FUNCTION VALUE
      -.10000000E+02  -.26305482E+01
      -.67745752E+01  -.12031250E+02
      -.17255491E+01  -.94947062E+01
      -.49138370E+00  -.12031250E+02
      .45576849E+01   -.94947062E+01
      .57918034E+01   -.12031250E+02

      NUMBER OF FUNCTION EVALUATIONS L =    128

      IFAIL = 0.  TERMINATION CRITERION: PROBABILITY OF FINDING THE
      GLOBAL OPTIMUM WITH GIVEN ACCURACY IS MORE THAN 0.95

      EXTR TERMINATED
```

Figure 18.14 Example of EXTR output

where the input is: X,A,B,N,IPAR,PAR,IPA,IPAA
 and the output is: X,FM X is initial point as input, and optimal point as output
 A is beginnings of optimization intervals
 B is end of these intervals
 N is dimensionality of X
 FM is optimal value
 IPAR(1)=IPR is a printing parameter
 IPAR(2)=M is the maximal number of function evaluation, $M \leq 500$.
 IPAR(3)=LT is the number of initial points which are random uniformly distributed points, $LT \geq 6$, recommended $LT = 6$.
 IPAR(4) is number of cycles
 IPAR(5) number of coordinate starting optimization in the first cycle
 PAR(1)=EPS1 is the accuracy of FM
 PAR(1+I)=EPS2 is the accuracy of X(I), $I=1,\dots,N$.

In the main program the following arrays should be described:
 IPAR(30), PAR(30).

EXAMPLE. Function: FURASN(X,N),
 Parameters: IPR=0, N=2, M=400, LT=6, EPS1=10⁶, EPS2=10⁶.

The main of EXKOR is in Figure 18.15.

The output is illustrated by Figure 18.16.

```

program exexkor
dimension x(2),a(2),b(2),ipar(30),par(30)
data n,nm,ipa,ipaa/2,400,0,0/,a/-0.25,-0.125/,b/0.5,0.625/
data ipar/0,100,6,2,1,25*0/
data par/0.01,0.01,0.01,27*0./
call exkor(x,a,b,n,fm,ipar,par,ipa,ipaa)
stop
end
function fi(x,n)
dimension x(n)
fi=furasn(x,n)
return
end

```

Figure 18.15 Example of EXKOR main

```

      E X K O R

      I N I T I A L   D A T A

      NUMBER OF VARIABLES           N =      2
      PRINTING PARAMETER           IPR =      0
      MAXIMUM NUMBER OF FUNCTION EVALUATIONS IN ONEDIMENSIONAL
      SEARCH                       M =    100
      NUMBER OF FUNCTION EVALUATIONS FOR PARAMETER ESTIMATION
      IN ONEDIMENSIONAL SEARCH     LT =      6
      NUMBER OF CYCLES              KC =      2
      NUMBER OF FIRST COORDINATE TO BE OPTIMIZED NO =      1

      ACCURACY OF OPTIMAL FUNCTION VALUE EPS1 = .99999998E-02
      ACCURACY OF OPTIMAL POINT
      .99999998E-02 .99999998E-02

      VECTOR OF LOWER BOUNDS (A) FOR X
      -.25000000E+00 -.12500000E+00
      VECTOR OF UPPER BOUNDS (B) FOR X
      .50000000E+00 .62500000E+00

      STARTING POINT
      .00000000E+01 .00000000E+01
      FUNCTION VALUE F = -.20000000E+01

      R E S U L T S

      OPTIMAL FUNCTION VALUE FM = -.20000000E+01
      OPTIMAL POINT
      .00000000E+01 .00000000E+01

      NUMBER OF FUNCTION EVALUATIONS L =    103

      EXKOR TERMINATED

```

Figure 18.16 Example of EXKOR output

18.3.9 MIVAR4: The local method of variable metrics by Tiesis

PURPOSE. To locate the local minimum of a differentiable function (18.1) defined on the rectangular parallelepiped (18.2).

RESTRICTIONS. Only the local minimum of differentiable functions can be found. Both a numerical and an analytical differentiation can be used. In the analytical case, the subroutine of differentiation should be provided.

ACCURACY. Arbitrarily close approaches to the minimum can be made depending on parameters EPS.

HOW TO USE THE METHOD

CALL MIVAR4 (X, A, B, N, HES, NH, FM, IPAR, IPA, IPAA)

where the input is: X, A, B, N, NX, IPAR, PAR, IPA, IPAA,
and the output is: X, FM.

HES is a working array of length NH which contains the elements of an inverse Hessian, where $NH=N(N+1)/2$.

In the main program the following one-dimensional array should be described:

IPAR(1)=IPR is a printing parameter
 IPAR(2)=M is the maximal number of function evaluation,
 IPAR(3)=NSTOP is the stopping parameter. If in the sequence of iterations of NSTOP length the values of function are decreasing less than EPS1 per iteration, then the procedure stops. Recommended NSTOP₁
 IPAR(4)=IMAX is the maximal number of iterations,
 PAR(1)=XEPS is the step length tolerance,
 PAR(2)=EPS is the norm of gradient tolerance,
 PAR(3)=EPS1 is the function decreasing tolerance,
 PAR(4)=DELTA is the length of the initial step of numerical differentiation.
 If partial derivatives are defined analytically, then the following subroutine is to be used to define a gradient:

```
SUBROUTINE GRABP1(X, N, A,B)
  DIMENSION X(N), A(N), B(N)
  COMMON/B3/GR(100)
```

```
      DO 1 I=1, N
1     GR(I)=∂f/∂xI
      RETURN
      END
```

The values of the gradient are stored in the one-dimensional array GR of length 100 by the common block /B3/.

EXAMPLE. The program locates the local minimum of the function (18.3). The parameters are: IPR=0, M=100, NSTOP=2, IMAX=100, XEPS=100, EPS=10⁻⁴, EPS1=10⁻⁴, DELT=10⁻⁴. The coordinates of initial point are (-0.1; 0.1).

The main of MIVAR4 is in Figure 18.17 and the output is illustrated by Figure 18.18.

```
      program exmivar4
      dimension x(2),a(2),b(2),hes(3),ipar(30),par(30)
      data n,nm,nh,ipa,ipaa/2,200,3,0,0/,a/-0.25,-0.125
      data b/0.5,0.625/,x/-0.1,0.1/
      data ipar/0,100,2,100,26*0/,par/100.,3*1.e-4,26*0./
      call mivar4(x,a,b,n,hes,nh,fm,ipar,par,ipa,ipaa)
      stop
      end
      function fi(x,n)
      dimension x(n)
      fi=furasn(x,n)
      return
      end
```

Figure 18.17 Example of MIVAR4 main

```

      M I V A R 4

      I N I T I A L   D A T A

      NUMBER OF VARIABLES                N =      2

      PRINTING PARAMETER                  IPR =      0
      MAXIMUM NUMBER OF FUNCTION EVALUATIONS  M =    100
      NUMBER OF SMALL FUNCTION CHANGE
      RECURRENCE NSTOP =      2
      MAXIMUM NUMBER OF ITERATIONS  IMAX =    100
      SMALL STEP TOLERANCE      XEPS = .10000000E+03
      GRADIENT NORM TOLERANCE    EPS =
      .99999997E-04
      FUNCTION CHANGE TOLERANCE EPS1 = .99999997E-04
      DIFFERENTIATION STEP      DELT = .99999997E-04
      VECTOR OF LOWER BOUNDS (A) FOR X
      -.25000000E+00  -.12500000E+00
      VECTOR OF UPPER BOUNDS (B) FOR X
      .50000000E+00   .62500000E+00

      STARTING POINT  -.10000000E+00   .10000000E+00
      FUNCTION VALUE F = .47440425E+00

      R E S U L T S

      OPTIMAL FUNCTION VALUE FM =  -.20000000E+01
      NORM OF CONSTRAINED GRADIENT = .12782573E-03
      OPTIMAL POINT  -.64514052E-06   .32752268E-06
      NUMBER OF FUNCTION EVALUATIONS L =    34
      NUMBER OF ITERATIONS NR =      5
      IFAIL = 1.  TERMINATION CRITERION:  CHANGE OF FUNCTION,
      LESS THAN EPS1, OCCURED NSTOP TIMES

      MIVAR4 TERMINATED

```

Figure 18.18 Example of MIVAR4 output

18.3.10 REQP: The local method of recursive quadratic programming by Biggs

PURPOSE. To locate the local minimum of a differentiable function (18.1) with non-linear constraints

RESTRICTIONS. Only the local minimum of differentiable functions with non-linear constraints can be found. Both numerical and analytical differentiation can be used. In the analytical case the subroutine of differentiation should be provided. The user should also provide the subroutine of constraints.

ACCURACY. Arbitrarily close approaches to the minimum can be made depending on parameters EPS.

HOW TO USE THE METHOD

CALL REQP (X, H, Q, GC, N, FM, IPAR, IPA, IPAA)

where the input is: X, N, IPAR, PAR, IPA, IPAA,
and the output is: X, FM.

The working arrays are: H, Q, GC

In the main program the following arrays should be described:

X(N), Q(N,N), H(N,N), GC(100, N), IPAR(30), PAR(30).

N ≤ 100

IPAR(1)=IPR is a printing parameter

IPAR(2)=IMAX is the maximal number of iterations,

IPAR(3)=NC is the number of equality constraints,

IPAR(4)=NIC is the number of inequality constraints,

PAR(1)=R1 is the penalty parameter, recommended R1=1.

PAR(2)=scale is a scale parameter of the penalty function,

recommended $0.1 \leq \text{SCALE} \leq 0.75$,

PAR(3)=DELTA is the step length of numerical differentiation,

recommended $10^2 \geq \text{DELTA} \geq 10^{-6}$

PAR(4)=EPS is the accuracy parameter, recommended $10^{-2} \geq \text{EPS} \geq 10^{-6}$

Constraints should be represented by the subroutine

CONSTR(X, N, G, MC)

where G is a one-dimensional array of length MC which contains the constraints at the point X , MC is the number of constraints.

The equality constraints should be represented at the beginning of array G . In the case of analytical differentiation the derivatives should be represented by the subroutine.

```

SUBROUTINE CALGRD(X, N, MC, GC)
COMMON/B3/GR(100)
DIMENSION X(N), GC(100, N), G(MC)
DO 1 I=1, N
GR(I)=∂f/∂xI
DO 1 J=1, MC
1 GC(J,I)=∂gj/∂xi
RETURN
END

```

The values of a gradient are stored in the one-dimensional array GR of length 100 by the common block $/B3/$. The values of gradients of constraints are put in the two-dimensional array GC of dimension $100*N$ in accordance with the following formula

$$GC(J,I) = \partial g_j / \partial x_i$$

where g_j is the constraint j .

EXAMPLE. The program locates the local minimum of the function

$$f(x) = 4x_1 - x_2^2 - 12$$

with constraints

$$\begin{aligned}
25 - x_1^2 - x_2^2 &= 0 \\
10x_1 - x_1^2 + 10x_2 - x_2^2 - 34 &\geq 0 \\
x_1 &\geq 0 \\
x_2 &\geq 0.
\end{aligned}$$

IPR=0, IMAX=50, R1=1, SCALE=0.25, DELTA=10⁻⁴, EPS=10⁻⁴,
NC=1, NIC=3. The initial point is (1,1)

The main of REQP is in Figure 18.19. The output is illustrated by Figure 18.20.

```
program exreqp
dimension x(2), ipar(30), par(30), h(2,2), q(2,2), gc(100,2)
data n, ipa, ipaa/2, 0, 0/, x/0.5, 0.5/
data ipar/0, 50, 1, 3, 26*0/, par/1., 0.25, 2*1.e-4, 26*0./
call reqp(x, h, q, g, n, fm, ipar, par, ipa, ipaa)
stop
end
function fi(x, n)
dimension x(n)
fi=4.*x(1)-x(2)**2-12.
return
end
subroutine constr(x, n, g, m)
dimension x(n), g(m)
g(1)=25.*x(1)**2-x(2)**2
g(2)=10.*x(1)-x(1)**2+10.*x(2)-x(2)**2-34.
g(3)=x(1)
g(4)=x(2)
return
end
```

Figure 18.19 Example of REQP main

```

      R E Q P

I N I T I A L   D A T A

NUMBER OF VARIABLES           N =      2
PRINTING PARAMETER           IPR =      0
MAXIMUM NUMBER OF ITERATIONS  IMAX =     50
NUMBER OF EQUALITY CONSTRAINTS  NC =      1
NUMBER OF INEQUALITY CONSTRAINTS NIC =     3
PENALTY PARAMETER            R1 = .10000000E+01
SCALING PARAMETER            SCALE = .25000000E+00
DIFFERENTIATION STEP DELTA = .99999997E-04
TOLERANCE LEVEL              EPS = .99999997E-04
STARTING POINT = INFEASIBLE
.50000000E+00   .50000000E+00
FUNCTION VALUE F = -.10250000E+02
CONSTRAINTS
.60000000E+01   -.24500000E+02
.50000000E+00   .50000000E+00

R E S U L T S

OPTIMAL FUNCTION VALUE FM = -.49521263E+02
OPTIMAL POINT   .13077012E+01   .65385065E+01
CONSTRAINTS
-.46767041E-05   -.72481351E-04
.13077012E+01   .65385065E+01
LAGRANGE MULTIPLIERS
-.39409199E-02   .42631583E+01
.00000000E+01   .00000000E+01
NUMBER OF ITERATIONS           K =     27
NUMBER OF FUNCTION EVALUATIONS  L =    113
NUMBER OF GRADIENT EVALUATIONS LG =     27
NUMBER OF ACTIVE CONSTRAINTS    MA =      2
NUMBERS OF ACTIVE CONSTRAINTS :      1      2
IFAIL = 0. TERMINATION CRITERION: NORMS
OF GRADIENTS LESS THAN EPS
      REQ P TERMINATED

```

Figure 18.20 Example of REQ P output

18.3.11 FLEXI: The local simplex method by Nelder and Mead

PURPOSE. To locate the local minimum of a differentiable function (18.1) with non-linear constraints

RESTRICTIONS. Only the local minimum of a function with constraints can be found. The user should provide a subroutine of constraints.

ACCURACY. Convergence to the minimum is not provided but usually the accuracy satisfies practical needs, if the number of iterations is large enough.

HOW TO USE THE METHOD

CALL FLEXI (X, H, FM, IPAR, IPA, IPAA)

where the input is: X, N, IPAR, PAR, IPA, IPAA,
and the output is: X, FM.

In the main program the following arrays should be described:

X(N), IPAR(30), PAR(30).

$N \leq 20$

IPAR(1)=IPR is a printing parameter

IPAR(2)=M is the maximal number of function evaluations,

IPAR(3)=NC is the number of equality constraints,

IPAR(4)=NIC is the number of inequality constraints, $NC + NIC \leq 100$.

PAR(1)=DELTA is the DIMENSION OF THE INITIAL SIMPLEX.

Recommended DELTA = $0.2\min(B(I) - A(I))$

PAR(2)=EPS is the stopping accuracy, recommended EPS = 10^{-5} or EPS = 10^{-6} .

Constraints should be represented by a subroutine

CONSTR(X, N, G, MC)

where G is a one-dimensional array of length MC which contains the values of constraints at the point X, and MC is the number of constraints.

The equality constraints should be represented at the beginning of array G.

EXAMPLE. The program locates the local minimum of the function

$$f(x) = 4x_1 - x_2^2 - 12$$

with constraints

$$\begin{aligned} 25 - x_1^2 - x_2^2 &= 0 \\ 10x_1 - x_1^2 + 10x_2 - x_2^2 - 34 &\geq 0 \\ x_1 &\geq 0 \\ x_2 &\geq 0. \end{aligned}$$

IPR=0, M=200, NC=1, NIC=3.

DELTA=0.3, EPS= 10^{-5} . The initial point is (1,1)

The main of FLEXI is in Figure 18.21 and the output is illustrated by Figure 18.22.

```

program exflexi
dimension x(2), ipar(30), par(30)
data n, ipa, ipaa/2, 0, 0/, x/0.5, 0.5/
data ipar/0, 50, 1, 3, 26*0/, par/0.3, 1.e-5, 28*0./
call flexi(x, n, fm, ipar, par, ipa, ipaa)
stop
end
function fi(x, n)
dimension x(n)
fi=4.*x(1)-x(2)**2-12.
return
end
subroutine constr(x, n, g, m)
dimension x(n), g(m)
g(1)=25.*x(1)**2-x(2)**2
g(2)=10.*x(1)-x(1)**2+10.*x(2)-x(2)**2-34.
g(3)=x(1)
g(4)=x(2)
return
end

```

Figure 18.21 Example of FLEXI main

```

      F L E X I

      I N I T I A L   D A T A

      NUMBER OF VARIABLES           N =    2
      PRINTING PARAMETER           IPR =    0
      MAXIMUM NUMBER OF FUNCTION EVALUATIONS   M =   50
      NUMBER OF EQUALITY CONSTRAINTS       NC =    1
      NUMBER OF INEQUALITY CONSTRAINTS      NIC =    3
      SIZE OF INITIAL POLYHEDRON DELTA =  .30000001E+00
      DESIRED CONVERGENCE           EPS =  .99999997E-05
      STARTING POINT   .50000000E+00   .50000000E+00
      FUNCTION VALUE F =  -.10250000E+02
      CONSTRAINTS
        .60000000E+01  -.24500000E+02
        .50000000E+00  .50000000E+00
      STARTING POINT - INFEASIBLE
      CALCULATED FEASIBLE STARTING POINT
        .10619404E+01  .52252302E+01
      FUNCTION VALUE F =  -.35055267E+02
      CONSTRAINTS
        .88990617E+00  .44095817E+00
        .10619404E+01  .52252302E+01

      R E S U L T S

      OPTIMAL FUNCTION VALUE FM =  -.49522427E+02
      OPTIMAL POINT   .13077219E+01   .65386019E+01
      CONSTRAINTS
        .96653355E-04  -.21364361E-03
        .13077219E+01  .65386019E+01
      NUMBER OF FUNCTION EVALUATIONS L =    45
      NUMBER OF ITERATIONS NR =    18
      IFAIL = 1. TERMINATION CRITERION:
      CONSTRAINTS ARE VIOLATED
      MORE THAN TOLERANCE CRITERION

      FLEXI TERMINATED

```

Figure 18.22 Example of FLEXI output

18.3.12 LBAYES: The local Bayesian method by Mockus

PURPOSE. To locate the minimum of a uni-modal function with noise on the rectangular parallelepiped (18.2).

RESTRICTIONS. Only the minimum of a uni-modal function is found.

ACCURACY. Arbitrarily close approaches to the minimum can be found with probability 1, when the number of iterations is large enough. The Bayesian step length provides the minimal average deviation in accordance with a given statistical model (see [100]).

The number of iterations should be sharply increased, if we wish to make the average error considerably less than the level of noise.

HOW TO USE THE METHOD

CALL LBAYES (X, A, B, N, F, IPAR, PAR, IPA, IPAA)

where the input is: A, B, N, IPAR, PAR, IPA, IPAA,
and the output is: X, F, XM, FM

where

X is the last point,

F is the value of the function at point X,

XM is the array of length N which defines the point of minimum of the function,

FM is the minimum

Arrays XM and FM are defined by the common block

COMMON/LAIK/FM, XM(100)

In the main program the following arrays should be defined:

A(N), B(N), IPAR(30), PAR(30), X(N),

N ≤ 100

IPAR(1)=IPR is a printing parameter

IPAR(2)=M is the maximal number of iterations,

IPAR(3)=NIPA is the number of integer variables. They should be at the beginning of the array X,

PAR(1)=ANIU is the rate of decreasing of the differentiation step,

PAR(2)=BETA is the rate of decreasing of the iteration step,
recommended BETA=1.-2*ANIU, ANIU=0.01 to 0.1.

In terms of expression (7.3.31) ANIU = ν , BETA = $1 - \nu - \alpha$, to provide the convergence $\alpha \geq 0$, $\nu > 0$, $\alpha + \nu < 0.5$, $\nu - \alpha > 0$.

EXAMPLE. The program locates the local minimum of the following function with noise

$$f(x) = \sum_{i=1}^n (x_i^3/6000 + x_i^2/200) + \xi$$

with $n = 2$, $x \in [-10, 10]$.

Here IPR=0, M=20, NIP=1, ANIU=0.05, BETA=0.9, and ξ is the random variable uniformly distributed in the interval $[-0.5, 0.5] * 0.046\sqrt{(n/2)}$, by the real function ATS(1) from this package. The initial point is (5,5).

The main of LBAYES is in Figure 18.23 and the output is illustrated by Figure 18.24.

```

program exlbayes
dimension x(2),a(2),b(2),ipar(30),par(30)
data n,ipa,ipaa/2,0,0/,a/-10.,-10./,b/10.,10./,x/5.,5./
data ipar/0,20,1,27*0/,par/0.05,0.9,28*0/
call lbayes(x,a,b,n,fm,ipar,par,ipa,ipaa)
stop
end
function fi(x,n)
dimension x(n)
fi=x(1)**3-12*x(1)*x(2)+8*x(2)**3+ats(1)
return
end

```

Figure 18.23 Example of LBAYES main

```

      L B A Y E S

      I N I T I A L   D A T A

      NUMBER OF VARIABLES           N =      2

      PRINTING PARAMETER           IPR =      0
      NUMBER OF ITERATIONS         M =     20
      NUMBER OF INTEGER VARIABLES NIPA =      1

      RATE OF TRIAL STEP DECREASING ANIU =
      .50000001E-01
      RATE OF ITERATION STEP DECREASING BETA =
      .89999998E+00

      VECTOR OF LOWER BOUNDS (A) FOR X
      -.10000000E+02  -.10000000E+02
      VECTOR OF UPPER BOUNDS (B) FOR X
      .10000000E+02   .10000000E+02

      STARTING POINT
      .50000000E+01   .50000000E+01
      FUNCTION VALUE F =   .82546283E+03

      R E S U L T S

      LAST FUNCTION VALUE F =  -.15957556E+01
      LAST POINT
      .20122664E+01   .46329933E+00

      OPTIMAL FUNCTION VALUE FM =
      -.15970689E+01 OBTAINED IN NR =   17
      OPTIMAL POINT
      .20122664E+01   .46329933E+00

      NUMBER OF FUNCTION EVALUATIONS L =   394
      LBAYES TERMINATED

```

Figure 18.24 Example of LBAYES output

18.3.13 Portability routines

The routines IIMACH and RIMACH define the machine constants. Those routines correspond to the PORT subroutine library (see [45]). The following table is an example for PC AT and compatibles.

INTEGER FUNCTION I1MACH(I):

	<u>I/O unit numbers</u>	Constants
		for PC AT
I1MACH(1)	is the standard input unit,	5
I1MACH(2)	is the standard output unit,	6
I1MACH(3)	is the standard punch unit,	7
I1MACH(4)	is the standard error message unit,	6
	<u>Words</u>	
I1MACH(5)	is the number of bits per integer storage unit	32
I1MACH(6)	is the number of characters per integer storage unit,	4
	<u>Integers</u>	
I1MACH(7)=A	is the base,	2
I1MACH(8)=S	is the number of base A digits,	31
I1MACH(9)=A**S-1	is the largest magnitude,	2147483647
	<u>Floating point numbers</u>	
I1MACH(10)=B	is the base,	2
	<u>Single precision</u>	
I1MACH(11)=T	is the number of base B digits,	23
I1MACH(12)=EMIN	is the smallest exponent E,	-128
I1MACH(13)=EMAX	is the largest exponent E,	127
	<u>Double precision</u>	
I1MACH(14)=T	is the number of base B digits,	52
I1MACH(15)=EMIN	is the smallest exponent E,	-1024
I1MACH(16)=EMAX	is the largest exponent E,	1023

REAL FUNCTION R1MACH(I):

Single precision machine constants

R1MACH(1)			
=B**(EMIN-1)	is the smallest positive magnitude		0.30e-38
R1MACH(2)			
=B**EMAX*			
(1-B**(-T))	is the largest magnitude		0.17e+39
R1MACH(3)			
=B**(-T)	is the smallest relative spacing		0.119e-06
R1MACH(4)			
=B**(1-T)	is the largest relative spacing		0.230e-06
R1MACH(5)			
=LOG10(B)			0.301e+00

Double precision machine constants

DR1MACH(1)			
=B**(EMIN-1)	is the smallest positive magnitude		0.419d-308
DR1MACH(2)			
=B**EMAX*			
(1-b**(-T))	is the largest magnitude		0.167d+309
DR1MACH(3)			
=B**(-T)	is the smallest relative spacing		2.22d-16
DR1MACH(4)			
=B**(1-T)	is the largest relative spacing		4.44d-16
DR1MACH(5)			
=LOG10(B)			3.01d-01

The list of machine constants should be fixed by operator DATA in the same order as shown here.

19

SOFTWARE FOR CONTINUOUS GLOBAL OPTIMIZATION USING UNIX C++

19.1 USER'S REFERENCE

19.1.1 Introduction

The set of optimization algorithms and the rules how to use them are similar to that of the Fortran library, but not the same. Therefore we will present a short description of these algorithms in this chapter.

19.1.2 Requirements

- - C++ compiler, for example, GNU g++ compiler or any ANSI C++ compliant compiler,
- - X-Window system,
- - UNIX (LINUX, SunOS, Solaris, HP-UX, AIX, ULTRIX, or compatible)
- - experience in writing simple C, or C++ functions.

19.1.3 Purpose of GM

GM is designed for continuous multi-variate global optimization.

The user defines the optimization problem in terms of variables describing:

- the parameters that can be changed (defined as variables),
- the function of those variables (defined as objective),
- some constraints restricting a possible change of variables.

Formally:

$$\min f(x), \tag{19.1}$$

$$a \leq x \leq b, \tag{19.2}$$

$$g_i(x) \leq 0, i = 1, \dots, p, \tag{19.3}$$

$$g_i(x) = 0, i = p + 1, \dots, m. \tag{19.4}$$

Here

x is a vector of n variables,

$f(x)$ is an objective,

$a \leq x \leq b$ defines rectangular constraints

$g_i(x) \leq 0, i = 1, \dots, p$ defines inequality constraints

$g_i(x) = 0, i = p + 1, \dots, m$ defines equality constraints

The objective function may be deterministic or (for some methods) with "noise".

The optimization region may be defined by rectangular constraints (19.2) or (for some methods) by linear and non-linear constraints defined by expressions (19.3) or (19.4).

19.1.4 Installation

If you run LINUX then the GM software will just run (hopefully). If you use another UNIX version, then you should change the 'Makefile' accordingly. Installing GM one should

- copy the archive file 'gmc.tgz' by the LINUX command


```
> mcopy a: gmc.tgz
```
- extract GM files from the archive 'gmc.tgz' by the LINUX command


```
> tar -zxf gmc.tgz
```

The GM software is tested by a small group of people. Therefore the authors will do what they can fixing errors and eliminating inconveniences. Contact e-mail:

jonas.mockus@ktl.mii.lt
jonas@optimum.mii.lt
audris@bell-labs.com
mockus@ecn.purdue.edu

19.1.5 Initialization

- Define the objective as some function defined as *fi*. The function *fi* has two parameters: an array of values of each variable and the number of variables. The function *fi* returns the value of objective, given the values of variables.
- Define "non-interactive" linear or non-linear constraints as a function named *constr*. It has four parameters: an array of values of variables, the number of variables, an array of values of constraints, and the number of constraints. The function *constr* returns the values of constraints, given the values of variables.
 The user can also define rectangular constraints interactively, during the optimization.
- Describe an objective and constraints in a file named `fi.C`
 - the objective routine is


```
double fi (const double *xx, int dim)
```
 - the constraints routine is


```
void constr (const double *x, int n, double *g, int ng)
```
- Start optimize the problem described in the file `fi.C`:

- set X-window
- compile
 - > `rm fi.o`
 - > `make`
- run
 - > `./test`

19.1.6 Menu System

- The Main Menu:
Global, Local, Operations, Quit, Parameters, Results, Output
- Global Optimization Menu:
Bayes1, Mig1, Unt, Exkor, Glopt, Lpmin
- Local Optimization Menu:
Nlp, Flexi, Lbayes
By default the initial point for local optimization is the result of previous global optimization.
- Operations Menu:
Run, Stop, Exit
- Quit Command
- Parameter Box
Enter the parameters of methods and point O.K.
For recommendations how to choose parameters see Chapter 18 and [100]).
- Results Box:
Shows the best objective Y and the best variables X(I). If you prefer no changes then point to O.K.
- Output Menu:
 - Convergence, Projection, and Numeric Windows
 - Convergence window shows how the objective depends on the iteration number.
 - Projection windows show how the objective depends on the different variables. Enter the number of variable defining the projection.

- Numeric window shows the current values of objective and variables.
- Using Menu:
 - To see some "invisible" components of vector point to 'down' or 'up' arrows. The vector will scroll, if there are invisible components.
 - To edit a variable, touch BACKSPACE¹, then edit the variable and point to O.K.
 - Enter the Parameter Box immediately after selecting a method.

19.2 DESCRIPTION OF METHODS

For a conceptual description of methods see [100, 104], and Chapter 4 of this book.

Here is a short description:

19.2.1 Global Methods

- Bayes1 is the Bayesian method by J.Mockus.
 - Region: rectangular
 - Objective: continuous (possibly with "noise")
 - Convergence: to global minimum (in probability, if noisy)
 - Remarks:
for "expensive" objectives using not too many observations.
- Mig1 is for uniform Monte Carlo search.
 - Region: rectangular
 - Objective: general
 - Convergence: in probability
 - Remarks:
for inexpensive and irregular objectives using great number of observations.

¹One should use the 'xmodmap' command to adapt the keymap, if BACKSPACE key is not working.

- Unt is the extrapolation type method by A. Zilinskas.
 - Region: rectangular
 - Objective: continuous
 - Convergence: to global minimum
 - Remarks:
for expensive objectives using not too many observations.
- Exkor is the Bayesian coordinate line search method by A. Zilinskas.
 - Region: rectangular
 - Objective: continuous
 - Convergence: to global minimum along the search line.
 - Remarks:
for approximately "separable" objectives and also for preliminary exploration using the projection windows.
- Glopt is the clustering method by Torn.
 - Region: rectangular
 - Objective: continuous
 - Convergence: not provided
 - - Remarks:
works well in many practical problems with a moderate number of local minima
- Lpmin is the uniform deterministic search by I.Sobolj and G. Dzemyda.
 - Region: rectangular
 - Objective: general
 - Convergence: to global minimum.
 - Remarks:
for inexpensive objectives using many observations

19.2.2 Local Methods

- Nlp is the non-linear programming method by K. Schittkowski
 - Region: defined by linear and non-linear constraints

- Objective: differentiable
 - Convergence: to local minimum
- Flexi is the simplex method by Nelder, Mead and Himmelblau.
 - Region: defined by non-linear constraints
 - Objective: non-differentiable
 - Convergence: not provided
- Lbays is the method of stochastic approximation with Bayesian step size control by J.Mockus.
 - Region: rectangular
 - Objective: continuous with noise
 - Convergence: in probability to local minimum

19.3 HP-UNIX CLASS-ROOM VERSION OF GM

The HP-UNIX Class-Room Version is for a HP server and a class of X-terminal users. In the example server files are in the directory 'classroom/server' and X-terminal files are in the directory 'classroom/user'.

Installing server

- change to server directory
 - > `cd server`
- copy the file 'gmhp.tgz'
 - > `cp gmhp.tgz`
- extract the GM files
 - > `tar -zxf gmhp.tgz`

Installing user

- change to user directory
 - > `cd user`

- copy the file 'Makefile'
 > cp Makefile
- adapt 'Makefile' to your HP-UNIX system, if needed
- copy user objective 'fi.C'
 > cp fi.C
- compile
 > make
- run
 > ./test

Example of 'Makefile'

An illustrative example of 'Makefile' is in Figures 19.1 and 19.2. For complete file see enclosed disk 'classroom/user/Makefile'.

```
#Makefile
#
# compile by GNU make
#
# adapt auxiliary variable:
#
ORIGIN =/home/classroom/server/gmhp
VPATH = .:${ORIGIN}

# pathname to X11/*.h files
# if compilation errors are "Can not find include file ..." then
# try /usr/local/include or /usr/include/X11R5
#
# appendix:
#
# Complete path for 'include' files
# was '.'
#
INCDIR      = ${ORIGIN} # . # /usr/include

# C++ compiler. If error is "Can not load ..."
# (try CC or gcc instead of g++)
C++         =gcc      -Wall -I${INCDIR} -L${LIBDIR} -O

# pathname to libX11.a
# If error is "Can not find library ..."
# try /usr/local/lib or /usr/lib/X11R5
LIBDIR      = /usr/lib/X11
#LIBDIR=/usr/local/lib
CPP         = /lib/cpp
F77        = f77
LIBS       = /usr/lib/X11R5/libX11.sl -lm
```

Figure 19.1 Example of the file 'user/Makefile', Page 1

```

# appendix:
# was only SRC = $(wildcard *.C)
SRC1 = $(wildcard $(ORIGIN)/*.C)
SRC = $(SRC1) ./fi.C
OBJS = $(notdir $(SRC:.C=.o))
RES      = menu.res bayes1p.res nlpp.res var.res lbayesp.res \
          res.res num.res mig1p.res untp.res exkorp.res lpminp.res \
          gloptp.res flexip.res help.res

TARGET    = test
.SUFFIXES: .C .o .rc .res .f
.f.o:     $(F77) -c $*.f
.C.o:
$(C++) -c $*.C
.rc.res:
$(CPP) -P $(ORIGIN)/$*.rc > $*.res
all: $(RES) $(TARGET)
libgm.a : $(OBJS)
ar cr libgm.a $(OBJS); ranlib libgm.a

$(TARGET): $(ORIGIN)/libgm.a fi.o $(RES)
$(PURIFY) $(C++) -o $(TARGET) $< fi.o $(LIBS)

.PHONY: clean
clean:
rm -f *.o *.a

depend: $(SRC)
rm -f $@
$(C++) -MM $^ >> $@

# DO NOT DELETE THIS LINE -- make depend depends on it.

```

Figure 19.2 Example of the file 'user/Makefile', Page 2

20

EXAMPLES OF UNIX C++ SOFTWARE APPLICATIONS

20.1 INTRODUCTION

In the next two sections we apply the continuous global optimization UNIX C++ software GM to the minimization of squared residuals (6.11) of ARFIMA model (see Chapter 6) as a function of the parameters d and $b = (b_1, \dots, b_q)$ while the parameters $a = (a_1, \dots, a_p)$ are defined by the corresponding system of linear equations (6.12). LINUX 1.2.8 version is used.

In Section 20.2, GM is applied in the interactive mode to estimate unknown parameters d and b by global optimization using merely a part of available data $t \leq T_0 < T$. Here T denotes all the data, and T_0 defines that part.

We predict the exchange rates using these estimates and assuming the Gaussian residuals ϵ_t , $t > T_0$ of the ARFIMA model. The prediction is repeated many times and the average, the upper and the lower values are defined. The results are compared with actual data $t > T_0$.

In Section 20.3, GM is used as a library of global optimization subroutines. In this case the main program performs many ARFIMA predictions and compares the average results with that of random walk (RW). Here we omit the parameter d , because we did obtain $d = 0$ in most of the previous cases¹.

In Sections 20.4 and 20.5 it is explained how to define the data while applying GM to the optimization of BHA parameters using randomly generated flow-shop and knapsack problems.

¹In some cases we obtained the optimal value $d > 0$ (see the rial/\$ exchange rate before the Iranian revolution). In such cases, not only b but also the parameter d , is optimized.

The traditional test problem of Rastrigin (see [127] and Section 18) is in the file 'gmc/fitest.C'.

20.2 EXCHANGE RATES LONG-TERM PREDICTION USING THE ARFIMA MODEL AND GM IN INTERACTIVE MODE

20.2.1 Optimizing Parameters a , b , and d

Installing

- copy the archive file 'arfima.tgz' by the LINUX command


```
> mcopy a: arfima.tgz
```
- extract files from the archive 'arfima.tgz' by the LINUX command


```
> tar -zxf arfima.tgz
```

Changing directory to 'gmc'

```
> cd gmc
```

Updating the parameter file 'fitimeb.h'

An illustration how to define the data in the file 'gmc/fitimeb.h' is in Figure 20.1. Here the matrix c denotes a bilinear extension (see the matrix C in expression (6.19))

Updating the objective function file 'fi.C'

We see in Figure 20.2 how to define data in the file 'gmc/fi.C'. In this figure the parameter R is the truncation parameter, the number of non-zero components of sequence (6.4)².

²Note that here the meaning of R is not the same as in Figure 20.1.

```

#define P 10 /* number of parameters a */
#define Q 2 /* number of parameters b */
#define S 0 /* number of rows of matrix c */
#define R 0 /* number of columns of matrix c */
#define T 496 /* number of data entries in DATAFILE */
#define DATAFILE "exchr" /* data */
#define T0 (int)(T/2) /* Number of entries for estimation */
#define N_MC 100 /* number of Monte Carlo simulations */

```

Figure 20.1 Example of the file 'gmc/fitimeb.h'

```

#include "fi.h"
#include <math.h>
#include <stdio.h>
#define P 5 /* number of parameters a */
#define Q 2
#define T 491
#define R 10
#define DATAFILE "intclosing"
static double z[T], w[T], d[R], A[T][P], B[T], ma[P][P], mb[P] ;
static double a[P] ;
const double *b ;
int number_of_variables = Q+1 ;

```

Figure 20.2 Example of the file 'gmc/fi.C'

Compiling

```

> rm fi.o
> make

```

Using the GM X-window menu system

- set X-window
- run

- ```
> ./test
```
- set a mode: GLOBAL or LOCAL
  - set a method in the mode GLOBAL
    - Bayes1
    - Mig1
    - Unt
    - Exkor
    - Glopt
    - Lpmin
  - or set a method in the mode LOCAL
    - Nlp
    - Flexi
    - Lbayes
  - set parameters in the PARAMETERS box
  - set an output mode form in the OUTPUT menu
  - set an operation mode in the OPERATIONS menu
  - read the optimal parameters in the RESULTS box

The meaning of parameters is similar to that of the GM Fortran version (see Chapter 18). The output format and the set of operation modes are described in Section 19.1. Erase by touching 'BackSpace' key, move down by pointing to  $\nabla$ , and move up by pointing to  $\Delta$ . One points to the mark *O.K.* before proceeding further.

## 20.2.2 Plotting Sum of Squared Residuals as Parameter Functions

Change directory to 'arfima'

```
> cd arfima
```

Update the data file 'data.C'

```

#include "fi.h"
#include <math.h>
#include <stdio.h>
#define N 3 /* Number of variables */
#define NVAR 0 /* Number of non fixed parameter */
double param[N] = {
0.e-01,
0.e-01,
0.e-08} ;
double f (double d)
{
 param[NVAR] = d ;
 return fi (param , N) ;
}

main ()
{
 for (double d = -1 ; d<=1 ; d+= 0.022)
printf ("%e %e\n", d, f(d)) ;
}

```

**Figure 20.3** Example of the file 'arfima/data.C'

The example how to define data in the file 'arfima/data.C' is in Figure 20.3. We have to:

- define the number of parameters 'N'=q+1
- define the number of the variable parameter 'NVAR'
- define the optimal values of N-1 non-variable parameters 'double param[N]'
- define the bounds of the variable parameter d in the loop: double d = -2 ; d<sub>i</sub>= 1 ; d+= 0.01

**Updating objective file 'fitime.C'** Figure 20.4 shows how to define data in the file 'arfima/fitime.C' Here the meaning of *R* is the same as in Figure 20.2 We have to:

```

#include "fi.h"
#include <math.h>
#include <stdio.h>
#define P 5 /* number of parameters a */
#define Q 2 /* number of parameters b */
#define T 400 /* number of data entries */
#define R 10 /* number of non-zero d-components */
#define DATAFILE "exchr" /* data */
static double z[T], w[T], d[R], A[T][P], B[T], ma[P][P], mb[P] ;
static double a[P] ;
const double *b ;
int number_of_variables = Q+1 ;

```

**Figure 20.4** Example of the file 'arfima/fitime.C'

- define the number of a parameters 'P'=p,
- define the number of b parameters 'Q'=q,
- define the number of data entries 'T',
- define the number of non-zero d components 'R',
- define the data file 'DATAFILE', for example, DATAFILE= 'exchr' where 'exchr' means \$/£ rate

### Compiling and computing

```

> cc -o data fitime.C data.C -lm
data > data.out

```

### Plotting by 'Gnuplot'

```

> gnuplot
gnuplot > set data style line
gnuplot > plot 'data.out'
gnuplot > q

```

If you like the picture on the screen, then you may obtain a "tex" file, for example:

```
> gnuplot
gnuplot > set term latex
gnuplot > plot 'data.out' with lines
gnuplot > q
> latex data
> xdvi data
```

### 20.3 EXCHANGE RATES "ONE-STEP" PREDICTION USING THE ARMA MODEL AND GM IN BATCH MODE

#### Installing

- copy the archive file 'arma.tgz' by the LINUX command

```
> mcopy a: arma.tgz
```
- extract files from the archive 'arma.tgz' by the LINUX command

```
> tar -zxf arma.tgz
```

#### Change directory to 'arma'

```
> cd arma
```

#### Update the file 'fitimeb.h'<sup>3</sup>

An example how to define data in the file 'arma/fitimeb.h' is in Figure 20.5. We have to:

- define parameters P,Q,S,R,T,T0 and B and C bounds,
- define data file DATAFILE,
- chose the global and the local method
- define parameters of these methods

---

<sup>3</sup>Not the same 'fitimeb.h' as in the directory 'arfima'.

```

#define P 10 /* number of parameters a */
#define Q 2 /* number of parameters b */
#define S 0 /* number of rows of matrix c */
#define R 0 /* number of columns of matrix c */
#define T 460 /* number of data entries in DATAFILE */
#define DATAFILE "intclosing" /* data */

#define TO (int)(T/4) /* Number of entries for estimation */

#define MAX_B_BOUND 1
/* -MAX_B_BOUND <= b[i] <= MAX_B_BOUND */
#define MAX_C_BOUND 0.01
/* -MAX_B_BOUND <= c[i][j] <= MAX_B_BOUND */
#define LOCAL_METH EXKOR
#define GLOBAL_METH BAYES1
#define GLOPT_MAX_IT 1000 /* glopt IT */
#define GLOPT_LT 200 /* glopt LT */
#define GLOPT_MAXL 1000 /* glopt MAXL */

#define NLP_MAX_IT 40 /* nlp IT */
#define NLP_M 0 /* nlp M */
#define NLP_ME 0 /* nlp ME */

#define BAYES1_MAX_IT 20000 /* bayes1 IT */
#define BAYES1_LT 10000 /* bayes1 LT */

#define EXKOR_MAX_IT 2000 /* exkor IT */
#define EXKOR_INIT_POINTS 6 /* exkor LT */

```

Figure 20.5 Example of the file 'arma/fitimeb.h'

### Compiling:

#### general case

```

> rm main.o
> rm fi.o
> make onestep
> onestep > onestep.out

```

See the results are in the file 'onestep.out':

- the first column denotes the sum of squared errors of the ARMA model
- the second column stands for the sum of squared errors of the RW model
- the third column defines the difference of squared errors of ARMA and RW models
- the fourth column means errors of the ARMA model
- the fifth column indicates errors of the RW model

In the file 'params.out' the optimal values of the first components of  $a$  and  $b$  are stored. The prediction is performed many times. For the first prediction a global method is used. For subsequent predictions a local method is applied, as usual.

**special case**  $a_1 = 1$

```
> rm main.o
> rm fia1.o
> make onestep1
> onestep1 > onestep1.out
```

## 20.4 OPTIMIZING BHA PARAMETERS IN FLOW-SHOP PROBLEM

An example how to define data in the file 'gmc/fi\_flow.C' is in Figure 20.6.

## 20.5 OPTIMIZING BHA PARAMETERS IN KNAPSACK PROBLEM

An example how to define data in the file 'gmc/fi\_knap.C' is in Figure 20.7.

```
#include <stdio.h>
#include <math.h>
#include <malloc.h>
#include <string.h>
#include "task.h"
#define DIM 3
#define NUMBER_OF_JOBS 7
#define NUMBER_OF_MACHINES 40
#define NUMBER_OF_FCALC 100
#define FALSE 0
#define TRUE 1
int number_of_variables = DIM;
```

**Figure 20.6** Example of the file 'gmc/fi\_flow.C'

```
#include "fi.h"
#include "task.h"
int number_of_variables = 3;
#define NUMBER_OF_FCALC 1
#define NUMBER_OF_OBJECTS 500
static int number_of_objects = NUMBER_OF_OBJECTS;
static int object [NUMBER_OF_OBJECTS] ;
static double object_cost [NUMBER_OF_OBJECTS];
static double object_weight [NUMBER_OF_OBJECTS];
static double ratio [NUMBER_OF_OBJECTS];
static double total_weight;
static double normx [3];
static int init = 0;
```

**Figure 20.7** Example of the file 'gmc/fi\_knap.C'

**PART VII**

---

**VISUALIZATION**



# 21

---

## DYNAMIC VISUALIZATION IN MODELING AND OPTIMIZATION OF ILL DEFINED PROBLEMS: CASE STUDIES AND GENERALIZATIONS

### 21.1 INTRODUCTION

We consider visualization as a decision optimization tool in problems where the model and/or the objectives are not well defined. Four specific problems representing different degrees of determination are investigated.

The first problem concerns a smooth dynamic representation of data collected at fixed locations. In the example we want to minimize deviations from a constant temperature over space and time.

The second and third problems are a dynamic representation of observations in the form of averages over regions in space and time, and they are exemplified by epidemiological data. We are looking for spatial-temporal patterns that can suggest the most efficient ways of prevention and control.

The fourth problem may be referred to as visual indexing. We perform an exploratory analysis of a large collection of complex objects. The example is a dynamic index to a collection of 30,000 images. We search for the "most interesting" subsets of images via visual inspection of the index. In all cases we define appropriate techniques for visual representation. We describe the software and hardware. A videotape which displays the results is available.

For the description of well-defined global optimization problems see [63, 64]. An application of some elements of visualization in those problems was considered by [96]. A number of visualization techniques for well-defined optimization problems are considered in [72].

In real-life applications we often encounter ill-defined problems. In this book we define a problem as ill-defined if it cannot be accurately described in terms of finding the optimum of a function (as opposed to the conventional meaning of an ill-defined problem which originates in the area of integral equations). In such cases we expect to improve the mathematical model and/or the objective after inspecting the results of modeling and optimization.

We think that in ill-defined problems the visualization is an essential part of efficient problem solving. It seems that as the model and the objective specification becomes less precise, the visualization becomes more important. We use a dynamic visualization as a modeling and optimization tool in the problems where a complete formalization of the model and/or of the objectives is difficult or impossible.

The investigation starts from a relatively "well-defined" problem of minimization of spatial and temporal temperature deviations on a thin metal plate. Then we consider a quantity which varies as a smooth function in space and time but is measured as average values over a region in space-time. As an example, the incidence rate of the disease mumps is considered. A natural objective is to minimize the impact of the disease using the minimum amount of resources but, unfortunately, it is not well-defined.

There was a common subproblem in both examples, namely, to find the optimal smoothing function that represents the temperature distribution in the first example and the incidence rates in the second one. Finding the optimal function can be transformed into a scalar optimization problem by taking a scalar measure of deviation from the best function, e.g., integrated squared error, maximum absolute error. However, those scalar measures do not represent all the information about the difference between the best and the considered function. Visualization of this difference conveys more information and helps to select a more appropriate solution.

In the last example both, the model and the objective, are not well defined. We are studying a large collection of digital images. To facilitate the search for the "most interesting" subsets of images, we construct an index of images in the collection. The index utilizes a small copy of each image (a "thumbnail") to represent the full-size version. We consider various layout techniques, including the Peano curve, and investigate the ways of speeding-up the search.

In this chapter, our techniques are tested using real data: temperature measurements in the first example, disease incidence rates in the second and third example, and a large collection of images in the last one. We use these data as

”teaching” sets for our visualization techniques. Similar techniques, software, and hardware may also be used in the cases when the data is obtained via computer simulation.

In practical optimization problems a set of objective function arguments corresponds to parameters of the model that is being optimized. The visualization of that model in the optimization process can provide a valuable insight and might lead to a revision of the objective function itself (after all, the objective function is just a scalar representation of optimality for the model of interest).

A common theoretical problem in all cases is to define the best visualization techniques to facilitate a better formulation of the problem and of the objective. It is important to design specific software and hardware systems to process the data efficiently and to display the results. There are two ways of displaying the results: non-interactive (on a videotape) or interactive (on a computer screen). Each way has its advantages and disadvantages. Thus we consider both.

## **21.2 INTERACTIVE AND NON-INTERACTIVE DYNAMIC GRAPHICS**

### **21.2.1 Graphical Techniques**

Graphical methods are widely used for the presentation of results. Here we consider visualization methods as a solution tool rather than as a way of presenting results. The main advantage of visualization methods is a possibility to communicate great amounts of information (“one picture is worth a thousand words”) in a short time period.

In optimization problems the objective function is a scalar or (sometimes) a vector representation of the “quality” of a very complicated model. One scalar value is, often, an incomplete specification of a model in vector-optimization problems. We, usually, are not certain how to relate different components of a vector-valued objective function or which one of many different Pareto-optimal decisions to take in vector-optimization problems. Communicating more information about the model in the process of optimization can be effectively handled using visualization methods. As the numeric optimization process is

dynamic by nature so must be the visualization methods used in optimization problems.

If, on the other hand, it is assumed that all the information about the model is contained in the values of the objective function, another approach can be taken. In such a case, the visualization should be performed on the values of the objective function and its arguments, namely, given function values  $f^i$  at points  $x_1^i, \dots, x_n^i$  visualize a set of vectors  $(f^i, x_1^i, \dots, x_n^i)$ . This problem presents a serious challenge in any nontrivial case when  $n > 2$ . The visualization system described in the last example addresses this problem in a new way (see Subsection 21.6.5).

We consider interactive and non-interactive dynamic graphical displays in this book. The choice of interactive or non-interactive graphical display depends on two factors:

- the amount of computer time needed to generate a graphical display
- the amount of time a user is willing to wait for the display to be generated

As computers become faster the advantages of interactive display seem to be obvious. But modeling and visualization tools become more complicated too, consuming all the increase in computing power.

The first two examples of this part use non-interactive graphics. We discuss the new interactive dynamic graphics system in the third example. The fourth example is interactive except for the preprocessing stage.

### 21.2.2 Non-interactive Dynamic Graphics Equipment

A special equipment has to be obtained, if we wish to use non-interactive dynamic graphics systems conveniently and economically. Therefore a potential user of those systems needs information not only about the algorithms and software, but about the hardware, too.

A non-interactive dynamic graphical display (animation) can be created by recording successive video frames, which are replayed at the rate of thirty frames per second (the NTSC video system used in the United States and Japan). We

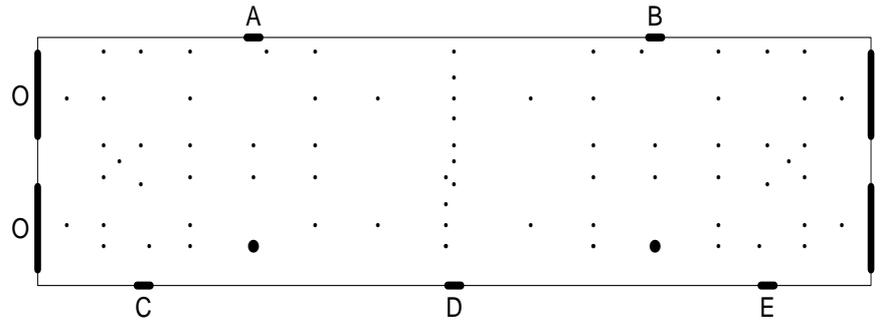
can play-back such recording on any standard home VHS video cassette recorder (VCR). We need a special video equipment (more sophisticated than a home VCR) to record individual video frames.

The generated images are recorded on a laser video disk recorder (LVR). Our model is Sony LVR-5000A. This recorder uses twelve-inch write-once disks. Each side of each video disk holds roughly 43,000 frames which is equivalent to about 23 minutes of dynamic graphics at 30 frames/sec. This laser video disk recorder has some special characteristics:

- One has an access to any individual frame almost instantaneously because the disk has a random access as compared to a sequential access on a one-tape device. Therefore LVR does not limit the recording speed.
- One can control LVR from the workstation over a standard RS-232 serial line using a simple protocol.
- One may jump instantaneously between arbitrary frames in the playback mode. This is an additional benefit of LVR, important for comparison. One can not do that with a tape.
- We can play back LVR at many different speeds both forward and backward.

We generate the images in a window of computer screen. We use a video scan converter (Otto Graphics converter Model 9500 produced by Folsom Research) to convert a high resolution component video (input of a workstation display) to the NTSC system.

The video equipment is completed with a VHS video cassette recorder, which is used to transfer animations onto conventional VHS videotapes, and a color monitor that can display both an RGB component video signal and an NTSC composite video signal.



**Figure 21.1** The metal plate showing locations of thermocouples

## 21.3 DYNAMIC GRAPHICS IN THE MANUFACTURING PROCESS

### 21.3.1 Background

Here the manufacturing process is described in general terms. The results in this section are adapted from [34] which contains more detailed description of the manufacturing process. The process of interest continuously generates a product. The product is divided into equal size units (completed during approximately thirty minutes of production time) to be further processed and shipped to the customer.

We are focusing on the spatial and temporal distribution of temperature over a thin rectangular metal plate which conducts the flow of the product via thousands of tiny holes. Figure 21.1 shows schematic of the plate as seen from the direction of product flow. The dots in the diagram indicate the locations of thermocouples used for gathering test data and will be described further. The holes which pass the product are not shown but are distributed nearly uniformly over the plate.

### 21.3.2 Objectives

Since the flow rate of the product is proportional to temperature (over a narrow range of operating temperatures), it is essential to have the temperature main-

tained constant (in time) and uniform (in space) over the surface of the plate. Consequently, a low variability of temperature in time increases the uniformity of the product, and low variability of temperature in space improves the process efficiency. If temperatures fall outside the operating range, the process fails to produce the desired product.

The temperature of the plate is affected by three main factors:

- The flow of product through the holes in the plate. The temperature of the product can vary.
- The flow of electric current across the plate. The flow of the current is controlled.
- The flow of coolant across the discharge surface of the plate.

Eddy and Shirakawa [37] described the spatial distribution of temperature over the plate by use of the heat equation (a partial differential equation) constrained by measured temperatures at the thermocouples (taking account both of the measurement error and incomplete specification of partial differential equations).

### 21.3.3 Data Collection

To gather the information the process engineers designed a special metal plate with 74 thermocouples at various locations on the plate. Subject to engineering constraints, the locations were chosen so that the distance from any point on the plate to the nearest thermocouple was small. Dots indicate the locations of the 74 thermocouples on the plate (the two large dots correspond to thermocouples used for the control of electric current) in Figure 21.1.

Data was gathered automatically during continuous operation of the process under various test conditions from the 74 thermocouples. Data was gathered for 300 time intervals corresponding to the production of a single unit of product. Such gathering process was repeated a number of times under different conditions.

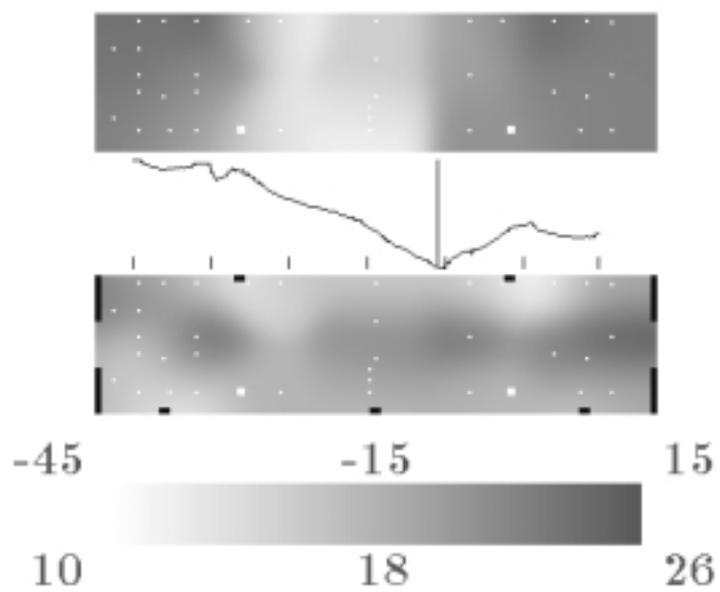


Figure 21.2 Heat distribution on the metal plate

### 21.3.4 Dynamic Display

An example of graphical display is given in Figure 21.2. Since the spatial variation in the data is ten times as great as the temporal variation at any fixed point in space, any dynamic display which does not remove the spatial variation would not reveal very much of the temporal variation. We combine two views of the plate: a static view showing a typical state and a dynamic view showing deviations from the typical state.

There are several regions in the display. The bottom rectangular region is static, and it displays median temperature across time. The top rectangular region displays the deviation from that median. The middle region represents a time series plot of the median temperature (over the thermocouples at each time point). Distance between tick marks corresponds to 50 time intervals (roughly 5 minutes). The thermocouples used for fitting are indicated in white and the electrodes and current taps are indicated in black. At bottom of the display a color scale gives correspondence between colors and temperatures. Two sets of numbers (below and above the scale) correspond to lower and upper images of the plate. The bottom numbers were linearly transformed to preserve confidentiality. Upper image of the plate shows residuals from median temperature and the numbers are given in degrees.

In the dynamic display, a vertical bar moves horizontally through the time series plot. The time location of the particular frame is indicated by the position of the bar.

We are convinced that the most important information in this example is conveyed by the dynamic displays. Also, it seems the most persuasive way to convey this information to the process engineers.

### 21.3.5 Results

The dynamic graphics had an immediate effect on the process engineers. They started several tests of various methods to control the coolant flow across the metal plate to reduce the spatial variation of the temperature. They also ran a number of tests to find the best locations for actual production process. As a result they decided to utilize a control system based on six thermocouples. Defining the optimal weights in the linear combination of the six thermocouples used to control the electric current is an optimization problem of some future.

We see that in this case optimization of the manufacturing process was indirect. The effect was achieved mainly by a better understanding of the system using the dynamic graphics<sup>1</sup>. We may apply the same dynamic graphic techniques for a direct optimization, too, using the data generated by computer simulation. It is a future task.

A direct optimization subproblem was to select a smoothing procedure such that the predicted temperatures would be close to the actual temperatures on the metal plate. The actual temperatures can be computed by solving heat transfer and electric current equations. Then we may optimize<sup>2</sup> the smoothing procedure. The work on this problem is ongoing. In the example we chose a smoothing parameter by selecting the visually acceptable dynamic display.

## 21.4 NON-INTERACTIVE DYNAMIC MAPS IN THE EPIDEMIOLOGICAL MODEL

### 21.4.1 Outline

Analysis of geographic data is important in many applications. Often, geographic data is reported as counts or averages over a region in space for an interval in time. We explore the intensity function underlying the data using dynamic graphics. We will refer to the dynamic display of such a function as a dynamic map. We think that this particular model (the intensity function) is, in many cases, most appropriate to visualize the spatial-temporal data. The aim of display of the intensity function is to detect spatial-temporal patterns.

We will exemplify this problem with data on the mumps disease collected monthly in the United States from 1968 until 1988. A brief summary of the problem and the results is given here. For a more detailed description of this example see [35].

Let us begin with a description of the problem we are trying to solve, namely, estimation and display of a smoothly varying function of space and time. We

---

<sup>1</sup>In our opinion, this is as legitimate a way of optimization as any other way

<sup>2</sup>We search for a smoothing technique which satisfies the appropriate physics equations as closely as possible

also provide a description of the data in a specific example under consideration (mumps).

We describe in some detail the animation methods used. Some statistical models are considered for estimating a smooth function from averages over regions. Two different animations with varying degrees of smoothness in space and time are described.

### 21.4.2 Objectives

Our objective is to estimate and display a smooth function with three arguments  $(x, y, t)$  representing two spatial coordinates  $(x, y)$  and time coordinate  $t$ . The available data consists of average values of the function over regions in space-time. A dynamic graphical display (animation) of the function consists of a sequence of images (frames) shown one after the other in a rapid sequence. Each image (frame) represents color map of the values of the function in the  $(x, y)$  plane. The sequence number of a frame represents the time dimension  $t$ .

### 21.4.3 Data

The mumps disease is of current public health interest in the United States in part because of a large outbreak which occurred in 1986-1987, primarily among unvaccinated adolescents and young adults in states without requirements for mumps vaccination.

The weekly provisional information on the occurrence of mumps (and some other diseases) is collected by the National Notifiable Diseases Surveillance System (NNDSS). Details concerning NNDSS can be found in [18]. The raw data consists of the number of cases of mumps reported from each state for each month for the period 1968-1988. Data is not available for some states and periods. One reason for this is that mumps has become reportable at differing times in the various states. Another reason is that small numbers of cases are less likely to be reported. We presume there are other reasons, too. The data-set spans 48 states times 21 years times 12 months per year which yields 12,096 possible state-month combinations. There are 1787 missing observations, approximately 15% of those possible.

The raw counts were converted to incidence rates for each state (cases per 100,000 population) by dividing by the estimated population in units of 100,000

people (we linearly interpolated 1970 and 1980 decennial census estimates of state population).

#### 21.4.4 Video Display

##### *Raw Data Animation*

The entire dataset consists of 252 months. After several experiments we decided that displaying the data at the rate of two-thirds of one second per month was a reasonable compromise between the time required to look at the entire data set and the apparent speed with which changes take place. In NTSC video (NTSC is the television signal used in the United States and Japan), 30 video frames (images) are shown in a second, so a month would be shown in 20 frames.

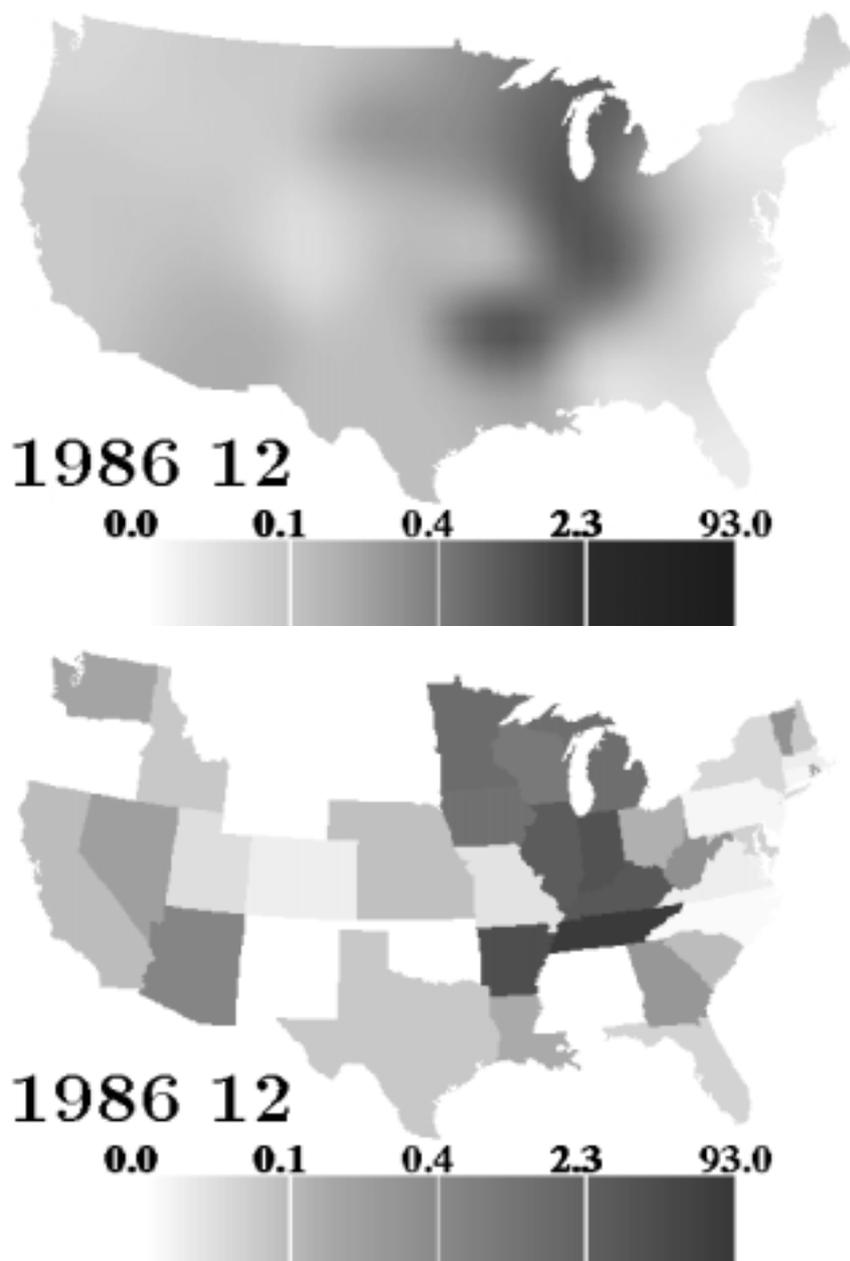
The speed of interactive animation would depend on the computer speed and the complexity of the animation algorithm, consequently the number of frames per month would vary accordingly. If the recording were done so that all identical frames were recorded and then the switch were made to the next month's data, the viewer would be distracted by the jumpiness of the resulting images.

Consequently, we choose to interpolate linearly between consecutive months. Precisely, the correctly colored maps for two consecutive months are calculated and then the intermediate maps are calculated by linear interpolation in the color scale. This produces a substantially smoother appearance.

##### *Smooth Animation*

It seems natural to assume that the incidence rate for a disease is actually given by a smooth function over the entire United States. The incidence rates computed for each state are the integral of this unknown function over the respective state (divided by the area of the state). Our problem then is to estimate the smooth function given its integral over the different states. A discussion on smoothing methods appropriate in this case can be found in [93, 35]. We sampled locations randomly in each state, assigned the state-month value to the sampled locations, and used a kernel smoothing technique to generate a map for each month.

As with the “raw” data, we interpolate linearly the intermediate frames between the monthly smoothed maps. Thus we smooth in space and in time using dif-



**Figure 21.3** Incidence Rates in December 1986. Smoothed Rates (top), Raw Rates (bottom)

ferent techniques. A single frame corresponding to December 1986 is displayed in Figure 21.3.

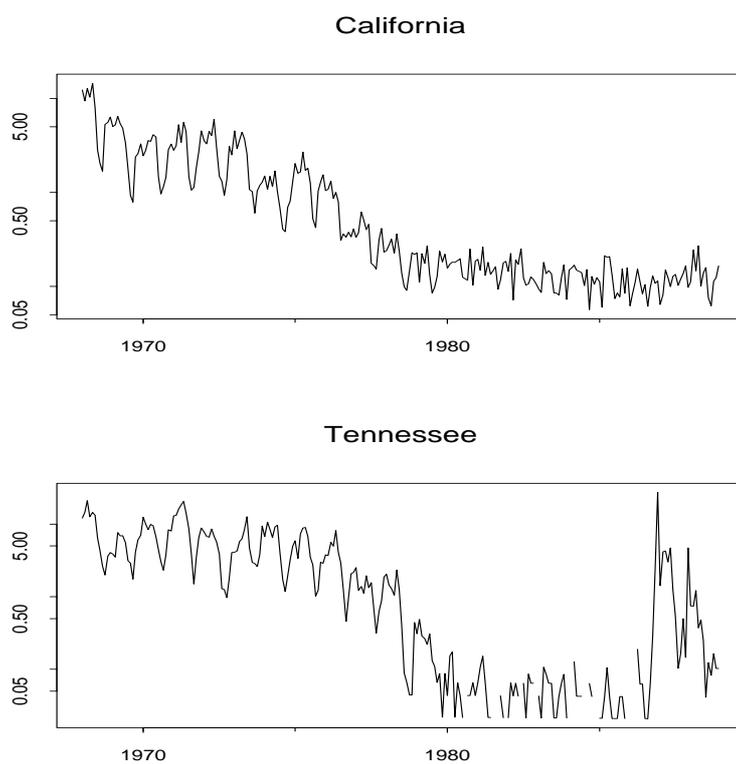
### 21.4.5 Effect of Dynamic Maps

The mumps disease has highest incidence before the vaccination started at the end of 1960's. In the 1970's vaccination almost completely wiped out the disease, leaving only a few cases per state per month. Mumps incidence rates in the US vary according to season. The peak occurs in early spring, while lowest incidence rates can be observed in autumn. School year could be an important factor since most of the cases are school age children.

The periodic effects are most obvious in the early years of the data set. Later, when the widespread use of the mumps vaccine reduced the typical monthly incidence rate below .1 cases per 100,000 people, the effect is not obvious in the raw version of the videotape. However, the smooth version still exhibits clear incidence differences between seasons.

The spatial spread of mumps can be seen in the raw data only after repeated viewing. It is most noticeable in the winter of 1987-1988 in the states surrounding Illinois. However, in the smoothed data the geographic spread of the disease is readily apparent. One can notice geographic areas that first reach high incidence rates in winter, and areas that are the last to be free from mumps in summer. Additional striking effect is relatively low (as compared to neighboring states) incidence rates in Louisiana in 1968-1972. This could reflect the fact that each state might use different disease reporting mechanism. The outbreak during the late winter of 1986-87 when the disease spreads from Illinois to Arkansas and Tennessee and in the subsequent winter when the disease spreads to all the neighboring states is more apparent in the smooth version of the videotape.

The vaccination programs were stopped in some states in the early 1980's and strong outbreaks of the disease occurred in 1986-1987 and in 1989, primarily among unvaccinated adolescents and young adults in the states without requirements for mumps vaccination. The explanation is well supported by the graph (see Figure 21.4) of the logarithm of incidence rates in California and Wisconsin. We can see a seasonal periodicity (high in spring and low in autumn) and an outbreak in Wisconsin in the second half of the eighties.



**Figure 21.4** Log of the mumps monthly incidence rates versus months from Jan. 1968 to December 1988

Clearly a non-interactive dynamic graphics system is useful for indirect optimization by filtering the general patterns. We need an interactive dynamic graphics system for the direct optimization.

## 21.5 INTERACTIVE DYNAMIC MAPS

### 21.5.1 Background

In this section we extend the results of the previous example to a more general setting. We consider an interactive dynamic map of the vector quantities that were aggregated over regions in space and time. Unlike the non-interactive case, for the interactive graphics we need a much larger set of potentially useful visualization and modeling tools.

As an example, 19 notifiable diseases are considered for spatial and temporal trends. The reported data are given on a state-by-month basis (for the period 1962-1992). We convert the reported cases to incidence rates before analyzing the data further. We produce interactive dynamic maps of those diseases trying to show the spatial and temporal behavior at the same time. The objective and the data in this case are very similar to the mumps example. There are two differences:

- multiple quantities (diseases) are visualized;
- an interactive approach is presented.

### 21.5.2 Interactions

The interactive tools can be divided into several classes according to their functionality. A brief discussion of each class follows.

#### *Transformations*

The usefulness of transformations in modeling is well known. The creation of a graphic display is a mapping from the data (or the model) to a range of display attribute values (e.g., range of pixels, colors, patterns, and glyph). To emphasize different features in the data or in the model one may want to use appropriate

transformations. In our system (it is currently under development), the user may choose different transformations: linear, general power, logarithmic, and rank.

### *Smoothing*

We need smoothing methods to predict the value at a point for a quantity given as an average over the region. It is easier to perceive the smooth model visually, especially with large amounts of information in a dynamic display. The region boundaries may contain no essential visual information and their display may hide important features of the model or of the data.

Different prediction techniques may be appropriate for different data/model combinations. A simple method is to use a constant value over the whole region. A disadvantage is jumps at the boundaries of the regions. An alternative method is to interpolate so that the interpolant has correct averages over the regions. Such interpolated values are more difficult for computing and may be outside the range of observations. We are currently adapting the Kriging method [93] and the Histospline method [32] to our system. The method described in [35] is currently being used.

### *Multiple Views*

We define the mapping from the model to a particular display as a view. Interactive selection of a view facilitates the visual inspection of different types of features present in the model and in the data.

A quantity of interest  $f$  is a function of three arguments  $x$ ,  $y$ , and  $t$  corresponding to latitude, longitude, and time. The dynamic display can be described as a function  $d_f$  of three variables: horizontal and vertical offsets  $d_x$  and  $d_y$  (describing the pixel location) and a frame number  $d_t$ . The range of possible values for  $d_f$  includes available colors, patterns, glyph, and the combinations of those named above. Hence, the view is a mapping from the quadruple  $(f, x, y, t)$  to  $(d_f, d_x, d_y, d_t)$ . The mapping with variable values of  $d_t$  would correspond to a dynamic map. When the view has a fixed value for  $d_t$  we obtain a static map. The view could be in the form of an  $XY$  plot, where, for example, the quantity  $f$  is mapped to  $d_y$  and  $t$  is mapped to  $d_x$ . In such a case, one could map the remaining arguments  $x, y$  to the frame number  $t$ . We have implemented three types of views corresponding to a dynamic map, to a static map, and to a time series plot.

### *Sections and Aggregation*

A view, as described in the previous section, is the mapping  $(f, x, y, t) \rightarrow (d_f, d_x, d_y, d_t)$ . A mapping that selects particular values of  $x, y$ , or  $t$  is a section, while the mapping that aggregates the values of  $f$  over subsets of  $x, y$ , or  $t$  is an aggregation. Section is a specific case of aggregation. For several quantities of interest  $f_1, \dots, f_n$ , the section may select one of those quantities, and aggregation may be a function on a subset of quantities.

We envisage four types of aggregation functions:

- Arithmetic (sum, variance).
- Order (minimum, maximum, median).
- Selection (section, several sections).
- Composition of all of the above mentioned.

### *Display of Vector Quantities*

In the mumps example we dealt with the incidence rates of one disease (mumps). If the incidence rates are reported for several diseases then one might be interested in detecting relationships between them. This raises the question of simultaneous display of multiple quantities that we try to address in our interactive dynamic graphics system.

We may use for vector quantities the following forms of display:

- side by side;
- alternate in time;
- use different attributes (color, pattern, height, transparency) for different quantities;
- use aggregation to produce a single quantity.

### *Display of Missing Values*

We implemented the following ways for handling missing values:

- a) leave out:
  - use neutral color;
  - use background color;
- b) fill in:
  - impute the value from available data (we use median for calculated value);
  - indicate that the value was imputed (we added a pattern to the calculated color).

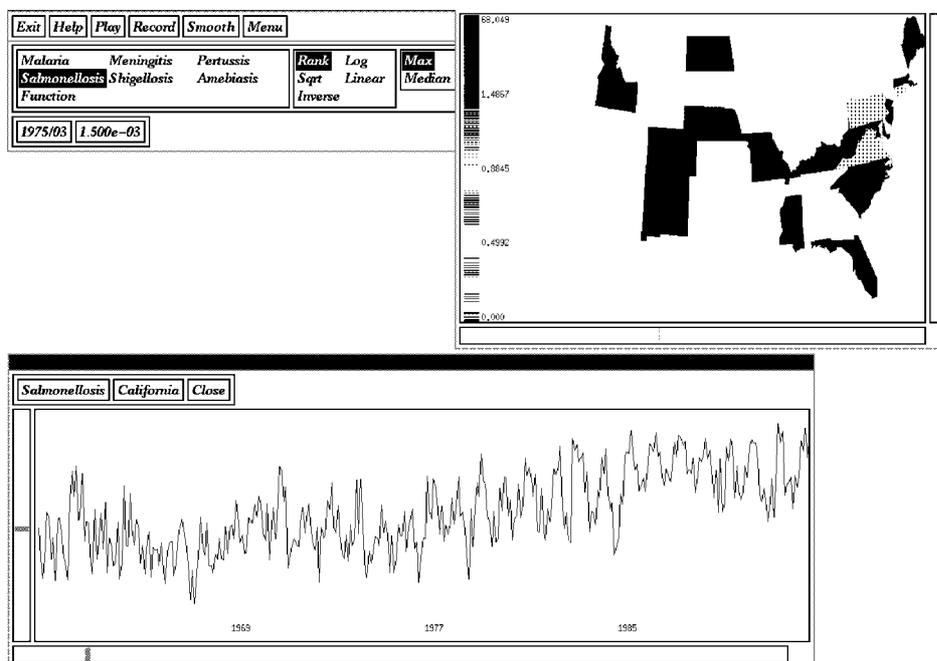
### 21.5.3 Display

An example display of our system is in Figure 21.5.

The system consists of the main control window and view windows. The control window contains menus and selection lists. Modeling and transformation methods are controlled from the main window. In Figure 21.5 the data on the disease salmonellosis is selected and the rank transformation is being used. The main window also contains the current date and time for the dynamic map view shown to the right. The bottom view contains a time series plot of disease incidence in California. The state and the disease can be selected interactively using scrollbars at the bottom and on the left of the time series plot.

### 21.5.4 Optimization Potential

We consider the interactive dynamic maps as an integral part of interactive optimization of various space-time systems, such as epidemiological, ecological, economic, social, public relation, etc., using the collected data and/or data from computer simulation of those systems.



**Figure 21.5** Control window and two views of the Interactive Map Animator

## 21.6 INTERACTIVE ICON INDEX

### 21.6.1 Outline

We are interested in interactive exploration of a large collection of complex objects, e.g, images, functions, and text, to name a few. Each object is regarded in the collection as an individual observation. To simplify the search for the "most interesting" object, we construct an index of objects in the collection. The index utilizes a small image (a "thumbnail") to represent each object. A large number of these thumb-nails are laid out in a workstation window. We refer to our system as an **I**nteractive **I**con **I**ndex, hence **I**<sup>3</sup>, hence, **I**ccube.

One can interactively arrange and rearrange the thumb-nails within the window. For example, one can order the thumb-nails by the values of a function computed from them or by the values of data associated with each of them. We access any individual object by simply clicking on the corresponding thumbnail. One selects subsets of objects by their attributes and values. Various operations on the selected subsets, e.g., animate, index, retrieve the original objects can be performed. We are currently extending the system to have a hierarchic index, a possibility to link different indices, and history functions to facilitate navigation.

The software may be regarded as the beginning of the development of exploratory tools for studying collections of complicated objects as we routinely study batches of numbers. Our focus to date has been on developing a tool that will assist in selecting the "best" individual objects or their subsets for detailed inspection. We anticipate that in the future it will be possible to consider summary information and distributional properties of various aspects of the objects, as well as a definition of optimal classification and search strategies.

The problem and the approach to its solution is described. The section is concluded with the description of a system optimizing layout strategies and list potential applications.

### 21.6.2 Objectives

An interactive dynamic index is designed that would be suitable for complex objects more general than text.

Term "object-base" is used, to refer to a structured collection of objects. In this way a data-base of general objects is distinguished from a data-base of

numerical and textual information. In the example an object-base containing approximately 30,000 images is considered.

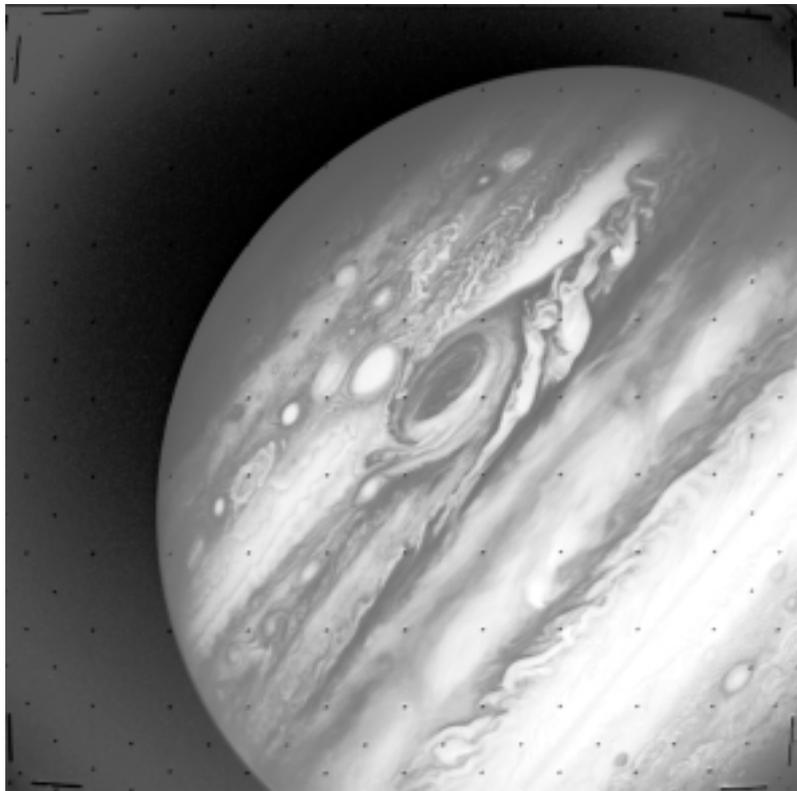
The techniques of search for interesting objects and groups of objects in the object-base are in development. Inspection and manipulation of individual objects is merely a part of this task. We are developing a hardware and software system to explore and to interact with the object-base. The system currently includes two workstation monitors, several CD-ROM players, and some specialized video equipment (a laser video disk recorder/player and a TV monitor). This system is regarded as a first tool for interactive exploratory analysis of this particular kind of a large object-base.

Additional goals are anticipated in the future, particularly the optimization ones, as our understanding of the object-base improves. A useful feature would be to cluster the images in the object-base and then use the images corresponding to the cluster centers in our index. If the number of cluster centers is much smaller than the number of images, one could inspect a very large object-base using a relatively small index.

### 21.6.3 Data

This example is described more in detail in [36]. The sample object-base was obtained from NASA's Voyager Project and Planetary Data System. The description here was derived from [38]. This collection contains images acquired by Voyager 1 and 2 as they passed by the planets Jupiter, Saturn, Uranus, and Neptune. The images show mentioned planets and their satellites. The archive consists of twelve CD-ROM volumes. Each volume contains approximately 2500 images stored in individual compressed files. For the description of the archive see [38]. A sample image from the collection is given in Figure 21.6.

Each object in our sample digital image archive is an image with associated attributes, such as information about what is in the image, a histogram of the pixel intensities, etc. Retrieving and sorting images by means of those attributes as keys can be accomplished as an enhancement of the standard data-base technology with the ability to handle images. Such attempts are usually referred to as multi-media data-bases (see [14]). We want to index images according to their appearance to better use the ability of the human visual system to detect relationships and unusual features.



**Figure 21.6** A sample image of Jupiter

### 21.6.4 Visual Representation and Interactions

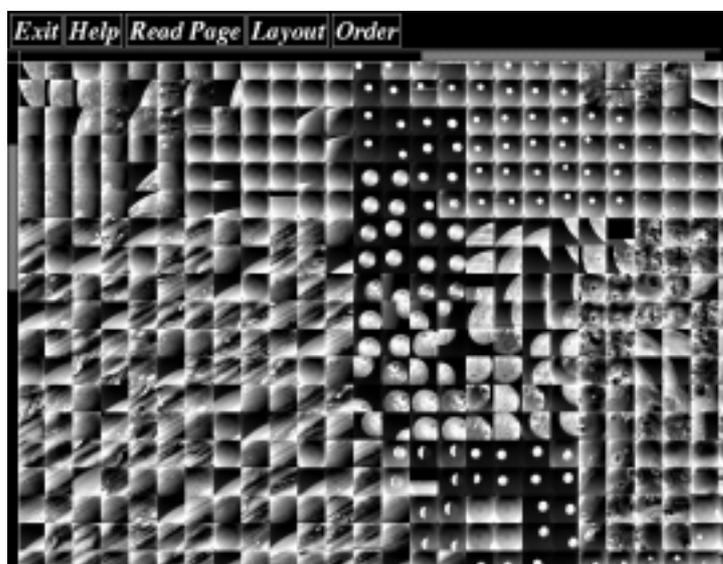
The main idea for the visual representation is simultaneous display and re-arrangement of all images in the collection. Since displaying images at full resolution would require a huge computer screen having the size of a football field, we achieve the simultaneous display by using small copies of the images (thumb-nails). We refer to the entire collection of thumb-nails as a *contact sheet* by analogy to photography. One can see from 775 to 7500 thumb-nails simultaneously in a window depending on thumbnail and window sizes (775 for 32 by 32 thumbnail in a 1000 by 800 contact sheet, 7500 for 16 by 16 thumb-nails in a 1600 by 1200 contact sheet). One can scroll the contact sheet to see all the images.

A very large collection of objects (more than 100,000) might exceed computer display resources and only thumb-nails corresponding to selected objects could be shown in a contact sheet. To select the objects we can classify them into a limited number of classes and only one thumbnail from each class would be shown in a contact sheet. Such higher level thumb-nails would serve as navigation tools to be expanded into the subset of thumb-nails corresponding to objects belonging to the same class. This expansion could be done in the same contact sheet or into another contact sheet by starting another **Icecube**. An inverse operation would be to collapse a selection of thumb-nails into one class and leave only one thumbnail as a representative from that class. The hierarchical indexing described in this paragraph is not implemented in **Icecube**.

A sample window of the Interactive Image Index appears in Figure 21.7.

Scrolling the contact sheet across the window is accomplished through standard scroll bars provided by the windowing system. Since our sample object-base could naturally be subdivided according to major planets, we were able to create separate pages (contact sheets) for each planet which could be handled more easily by our windowing system. Creation of an individual page can be done at the preprocessing stage once the object-base has been created.

The thumb-nails can be rearranged in the contact sheet by ordering the thumb-nails and by using different layout methods. In addition to display and re-arrangement we can select individual and subsets of thumb-nails for further inspection.



**Figure 21.7** A sample window of Interactive Image Index with a contact sheet containing thumb-nails of Jupiter

## *Ordering*

We order all the thumb-nails to facilitate a fast layout and selection. A simple layout (see “Layout” subsection) is a function giving a position in the contact sheet for each order number. The selection of thumb-nails by point-and-click requires an inverse function, namely, to identify a thumbnail located at a given position in the contact sheet. One may change the order interactively by sorting according to various keys (that represent image attributes) or by randomly assigning order numbers to each thumbnail. We have chosen to use a *stable* sort (see, [78]) in order to provide hierarchical sorting without sorting on multiple keys simultaneously.

## *Layout*

One can lay out the thumb-nails within the contact sheet in certain predetermined patterns. The natural patterns (similar to the text flow on paper) are across, from left to right, and down, from top to bottom. One can interactively specify the number of columns or rows to emphasize any possible periodicities in the sequence of thumb-nails. We found it useful to discover clusters of similar images. Simple layouts did not accomplish the goal of “keeping the neighbors”<sup>3</sup>. Thus, an capability of laying out the images along a space-filling curve was added. The “Peano” space-filling curve (see [121, 13]) was chosen.

## *Selection*

We record the images in standard video format on a Laser Video Recorder (see the “Non-interactive Dynamic Graphics Equipment” subsection) rather than digital format. The video disk recorder (as opposed to reading the digital image from a CD-ROM) allows a nearly instantaneous access. The user simply points and clicks on the desired thumbnail. This is the single most valuable feature of our system. The objects other than images might not be as convenient to display using video recording. Fast retrieval of a non-image type object might need other implementation. One may also select and play back any subsequence of images, or create a separate contact sheet from the selected thumb-nails.

---

<sup>3</sup>By “keeping the neighbors” we mean that images near in linear ordering will remain close on the screen, too.

### *Creation of Thumb-nails*

An important assumption of having the collection of thumb-nails as the object-bas index is a possibility of determining the object content from the thumbnail. When indexing a collection of images to create a thumbnail one has to reduce the original image in size without losing the image content (information that distinguishes the image from the other images in the collection). In our collection the small-scale features of images were not important, therefore pixel averages of the brightness-adjusted image were used to produce a thumbnail.

For the objects representing X-Y plots (e.g.,  $f(x)$ ), one can just scale a plot to fit into the thumbnail. For a scalar function of two arguments  $f(x, y)$  one encodes the values of a function by colors of the corresponding pixels in a thumbnail.

## 21.6.5 Results

### *Application Potential*

To appreciate the real application potential one must collaborate with the experts in the corresponding fields. Our experience lets us suppose that the system may be applied to such diverse areas as medical imaging, earth imaging, organic chemistry, astronomy, and large data-bases. In many scientific fields large amounts of data are being collected. It becomes more and more important to be able to retrieve relevant data. We have immediate plans to use the **Iccube** in at least two applications described below.

We are currently working on the analysis of large collections of functional magnetic resonance (fMRI) images of the brain obtained under various experimental conditions. A general purpose of the project is the localization of various cognitive functional areas of the human brain. Due to a small signal-to-noise ratio in those experiments, a large number of images is needed to show brain activation patterns. Imaging equipment can take approximately five images per second and an average experiment takes about one hour,

bringing the total number of images to about 18 thousand per experiment. We intend to use the **Iccube** to select and process relevant images from different fMRI experiments.

Another project is to use the **Iccube** for non-image data. We have obtained some data from the Centers for Disease Control on 19 notifiable diseases in the

United States (see sections “Non-interactive Dynamic Maps in Epidemiological Model” and “Interactive Dynamic Maps”). The data represents counts for 372 months and 48 continental states. In the **Icecube** one can consider this data as  $48 \times 19$  different time series of length 372, as  $48 \times 19 \times 31$  different time series of length 12 (corresponding to one year), or as  $372 \times 19$  different maps.

### *Icecube and Optimization*

A feasibility of the concept of visual indexing is demonstrated in the example. The effect of visual indexing can be appropriately estimated only by using the index while solving real problems. Therefore we merely mention some of the features which could be useful for optimization.

- visual classification;
- search for patterns;
- selection of subsets;
- extensibility.

The extension of **Icecube** to include new features valuable for image and object optimization <sup>4</sup> is under development.

One may apply the **Icecube** for visualization of some conventional optimization problems, too, using, for example, the following strategies:

- Take thumb-nails to represent a projection of the values of the objective function in a particular direction. Each thumbnail would correspond to a different direction.
- Take thumb-nails to represent objects that correspond to the argument values of the objective function.
- Take thumb-nails to represent each local minimum found in the optimization process.

---

<sup>4</sup>By image and object optimization we understand the search for the “most interesting” image(s) or object(s).

## 21.7 SUMMARY

The importance of visualization techniques in modeling and optimization of decisions were demonstrated by four different examples. That is a first step hoping that these techniques will be widely applied in global and discrete optimization, eventually.

Non-interactive visualization methods were considered in the first two examples and interactive methods in the remaining two. The first three examples describe attempts of using visualization when the objective is infinite dimensional, i.e., a function of three arguments (two-dimensions in space and time). In the first example, the objective was temperature distribution over a metal plate, in the second and third examples, the objective was disease incidence rates in the United States.

The last example presents a different approach in which the emphasis is laid on the ability of the human visual system to detect patterns. We simultaneously display a large number of “thumb-nails” (each representing a complex object) on a computer window and rearrange them to help classify and/or select the objects of interest.

In the considered cases, there was no obvious scalar or vector function that would describe the objective completely. One may use dynamic graphics as a natural, convenient and efficient way of direct interaction between the real or simulated data and a human decision maker. That is most important while solving ill defined optimization problems.



---

## REFERENCES

- [1] A. Alisauskas and V. Saltenis. On investigation of strategy of man while solving multi-ekstremal problems. In *Proceedings of VI-th Symposium on Cybernetics*, volume 4, pages 5–10, Tbilisi, Institut of Cybernetics, 1972.
- [2] M.R. Andenberg. *Cluster Analysis for Applications*. Academic Press, New York, 1973.
- [3] S. Andradottir. A global search method for discrete stochastic optimization. *SIAM Journal, Optimization*, 6:513–530, 1996.
- [4] I.P. Androulakis and V. Venkatasubramanian. A genetic algorithmic framework for process design and optimization. *Computers in Chemical Engineering*, 15:217–228, 1991.
- [5] Lee B. and G.V. Reklaitis. Optimal scheduling of batch processes for heat integration, part ii. *Computers & Chem. Eng.*, 1995. in print.
- [6] Kenneth R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York, 1974.
- [7] A. Baskis and L. Mockus. Application of global optimization software for the optimization of differential amplifier. In *Theory of Optimal Decisions*, pages 9–16. Vilnius, Lithuania, 1988. In Russian.
- [8] E. B. Baum and K. J. Lang. Constructing hidden units using examples and queries. In *Advances in Neural Information Processing Systems*, volume 3, pages 904–910. Morgan Kaufmann, San Mateo, CA, 1991, 1991.
- [9] L.N. Belykh. On the computational methods in disease models. In *Mathematical Modeling in Immunology and Medicine*, pages 79–84. North-Holland, Amsterdam, 1993.
- [10] L.T. Biegler, I.E. Grossmann, and G.V. Reklaitis. Applications of operations research methods in chemical engineering. In *Engineering Design: Better Results Through OR Methods*. North-Holland, Amsterdam, 1988.

- [11] M. C. Biggs. Constrained minimisation using recursive quadratic programming: some alternative subproblem formulations. In *Towards global optimisation*. North Holland, Amsterdam, 1975.
- [12] G. Box and G. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, CA, 1976.
- [13] A.R. Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Transactions on Computers*, C-20:424–426, 1971.
- [14] R.G.G. Cantell. Special issue on next-generation database systems. *Comm. ACM*, 34, 1991.
- [15] A. M. Chen and R. Hecht-Nielsen. On the geometry of feedforward neural network weight spaces. In *Proc. Second IEE International Conference On Neural Networks*, pages 1–4, London, 1991. IEE Press.
- [16] Yin-Wong Cheung. Long memory in foreign exchange rates. *Journal of Business and Economic Statistics*, 1:93–101, 1993.
- [17] Yin-Wong Cheung and K. Lai. Fractional co-integration analysis of purchasing power parity. *Journal of Business and Economic Statistics*, 1:103–112, 1993.
- [18] T.L. Chorba, R.L. Berkelman, S.K. Safford, N.P. Gibbs, and H.F. Hull. Mandatory reporting of infectious diseases by clinicians. *Journal of the American Medical Association*, 262:3018–3019, 1989.
- [19] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. In *Abstract, CMU Symposium on New Directions and Recent Results in Algorithms and Complexity*, 1976.
- [20] L.O. Chua and P.M. Lin. *Computer-Aided Analysis of Electronic Circuits*. Prentice-Hall, 1975.
- [21] Ching-Fan Chung. A note on calculating the autocovariances of the fractionally integrated arma models. *Economics Letters*, 45:293–297, 1994.
- [22] V. Chvatal. A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.
- [23] H. Cramer and M.R. Leadbetter. *Stationary and Related Stochastic Processes*. John Wiley, New York, 1967.
- [24] P. Craven and G. Wahba. Smoothing noisy data with spline functions. *Numerische Mathematik*, 31:377–403, 1979.

- [25] P.D. Crout. A short method for evaluating determinants and solving systems of linear equations with real and complex coefficients. *AIEE Transactions*, 69:1235–1241, 1941.
- [26] H. Das, P.T. Cummings, and M.D. LeVan. Scheduling of serial multi-product batch processes via simulated annealing. *Computers and Chemical Engineering*, 14:1351–1362, 1990.
- [27] M. DeGroot. *Optimal Statistical Decisions*. McGraw-Hill, New York, 1970.
- [28] P. Diaconis. Bayesian numerical analysis. In *Statistical Decision Theory and Related Topics*, pages 163–175. Springer Verlag, 1988.
- [29] F. X. Diebold and G. D. Rudebusch. Long memory and persistence in aggregate output. *Journal of Monetary Economics*, 24:189–209, 1989.
- [30] L.C.W. Dixon and G.P. Szego. *Towards global optimisation 2*. North Holland, Amsterdam, 1978.
- [31] R.A. Donnelly and J.W. Rogers. A discrete search technique for global optimization. *International Journal of Quantum Chemistry: Quantum Chemistry Symposium*, 22:507–513, 1988.
- [32] N. Dyn and G. Wahba. On the estimation of functions of several variables from aggregated data. *SIAM J. Math. Anal.*, 13:134–152, 1982.
- [33] G. Dzemyda and E. Senkiene. Simulated annealing for parameter grouping. In *Transactions. Information Theory, Statistical Decision Theory, Random Processes*, pages 373–383, Praque, 1990.
- [34] W.F. Eddy and A. Mockus. An example of noninteractive dynamic graphics for manufacturing process data. *International Statistical Review*, 61:81–95, 1993.
- [35] W.F. Eddy and A. Mockus. An example of the estimation and display of a smoothly varying function of time and space - the incidence of the disease mumps. *Journal of the American Society for Information Science*, 45(9):686–693, 1994.
- [36] W.F. Eddy and A. Mockus. An interactive image index. Technical Report 601, Department of Statistics, Carnegie Mellon University, 1994.
- [37] W.F. Eddy and K Shirakawa. Dynamic graphical presentation of high dimensional data for use in process control. In *Proceedings of the Conference on Statistics in Industry, Science and Technology*, pages 16–21, 1994.

- [38] E. Eliason, R. Davis, M. Martin, C. Avis, J. Hyon, B. Mehlman, and D. McMacken. *Archive of Digital Images from NASA's Voyager 1 and 2 Missions*. NASA, 1991.
- [39] Yu. Ermoljev and R.J-B. Wets. *Numerical Techniques for Stochastic Optimization*. Springer-Verlag, Berlin- New York-London, 1988.
- [40] Yu.G. Evtushenko. *Numerical optimization techniques*. Optimization software, Inc., New York, 1985.
- [41] F.Alufi-Pentini, V. Parisi, and F. Zirilli. Global optimization and stochastic differential equations. *J. of Optimization Theory and Applications*, 47:1–16, 1985.
- [42] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.
- [43] P. Floquet, P. Pibouleau, and S. Domenech. Scheduling and simulated annealing application to a semiconductor circuit fabrication plant. *Computers and Chemical Engineering*, 14:1351–1362, 1993.
- [44] C.A. Floudas and P.M. Pardalos. A collection of test problems for constrained global optimization algorithms. In *Lecture Notes in Computer Science, # 455*. Springer-Verlag, Berlin, 1987.
- [45] P.A. Fox, A.D. Hall, and N.L. Schryer. Bell laboratories computing science technical report. Technical Report 47, Bell Laboratories.
- [46] R. Fox and M. Taqqu. Large-sample properties of parameter estimates for strongly dependent stationary gaussian time series. *Annals of Statistics*, 14:517–532, 1986.
- [47] P. Freitas. Solving differential equations using event driven techniques. Technical Report ESPRIT1058/INESC/6.87/D8728, INESC, Lisboa, Portugal, 1987.
- [48] J.H. Friedman, M. Jacobson, and W. Stuetzle. Projection pursuit regression. Technical Report 146, Department of Statistics, Stanford University, March 1980.
- [49] C.B. Garcia and W.I. Zangwill. *Pathways to Solutions, Fixed Points, and Equilibria*. Prentice-Hall, Englewood Cliffs, 1981.
- [50] M.R. Garey and D.S. Johnson. Approximation algorithms for combinatorial problems: An annotated bibliography. In J.F.Traub, editor, *Algorithms and Complexity*, pages 41–52. Academic Press, New York, 1976.

- [51] J. Geweke and S. Porter-Hudak. The estimation and application of long memory time series models. *Journal of Time Series Analysis*, 4:221–238, 1983.
- [52] B.V. Gnedenko and I.N. Kovalenko. *Introduction to the Theory of Queueing*. Nauka, Moscow, 1987. in Russian.
- [53] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, 1989.
- [54] T. Gonzales and S. Sahni. Flow shop and job shop schedules. Technical report, University of Minnesota, Computer Science Technical Report, 1975.
- [55] D. Gorse, A. Shepherd, and J. G. Taylor. Avoiding local minima by progressive range expansion. In *Proceedings of the 1993 International Conference on Artificial Neural Networks*, volume 1, Amsterdam, the Netherlands, 1993.
- [56] C.W.J. Granger and R. Joyeux. An introduction to long-memory time series models and fractional differencing. *Journal of Time Series Analysis*, 1:15–39, 1980.
- [57] G.M. Guisewite and P.M.Pardalos. Minimum concave-cost network flow problems: Applications, complexity, and algorithms. *Ann. of Operations Research*, 25:75–100, 1990.
- [58] W. Hanson and K. Martin. Optimizing multinomial logit profit function. Technical report, Krannert School of Management, Purdue University, West Lafayette, IN, March 1994.
- [59] P. Helman, B.M.E. Moret, and H.D. Shapiro. An exact characterization of greedy structures. *SIAM Journal of Discrete Mathematics*, 6:274–283, 1993.
- [60] W. H. Highleyman. Linear decision functions with applications to pattern recognition. In *Proc. IRE - 50*, 1962.
- [61] Himmelblau. *Applied Nonlinear Programming*. McGraw-Hill, 1972.
- [62] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [63] R. Horst and P. Pardalos. *Handbook of Global Optimization*. Kluwer Academic Publishers, Dordrecht/Boston/London, 1995.
- [64] Reiner Horst, Panos M. Pardalos, and Nguyen V. Thoai. *Introduction to Global Optimization*. Kluwer Academic Publishers, Dordrecht, 1995.

- [65] J.R.M. Hosking. Fractional differencing. *Biometrika*, 68:165–176, 1981.
- [66] J.R.M. Hosking. Modeling persistence in hydrological time series using fractional differencing. *Water Resources Research*, 20:1898–1908, 1984.
- [67] D. R. Hush, B. Horne, and J. M. Salas. Error surfaces for multilayer perceptrons. In *IEEE Trans. on Systems, Man and Cybernetics*, volume 22(5), pages 1152–1161, 1992.
- [68] J. N. Hwang, S. R. Lay, M. Maechler, D. Martin, and J. Schimert. Regression modeling in backpropagation and projection pursuit learning. In *IEEE Transactions on Neural Networks*, volume 5(3), pages 342–353, May 1994.
- [69] O.H. Ibarra and C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of ACM*, 22:463–468, 1975.
- [70] G. Janacek. Determining the degree of differencing for time series via the long spectrum. *Journal of Time Series Analysis*, 3:177–188, 1982.
- [71] E. M. Johansson, F. U. Dowla, and D. M. Goodman. Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. *International Journal of Neural Systems*, 2(4):291–301, 1992.
- [72] Chris Jones. *Visualization and Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.
- [73] D.R. Jones, C.D. Perttunen, and B.E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Application*, 79:157–181, 1993.
- [74] R.H. Jones and A.V. Vecchia. Fitting continuous arma models to unequally spaced spatial data. *JASA*, 88:947–954, 1993.
- [75] J.B. Kadane and G.W. Wasilkowski. Average case  $\epsilon$ -complexity in computer science- a bayesian view. In *Bayesian Statistics 2*, pages 361–374. Elsevier Science Publishers, 1985.
- [76] V. Katkovnik. *Linear Estimates and Stochastic Problems of Optimization*. Nauka, Moscow, Russia, 1976. In Russian.
- [77] J. Kiefer. Sequential minimax search for a maximum. *Proceedings of American Mathematical Society*, 4:502–506, 1953.
- [78] D.E. Knuth. *The Art of Computer Programming, Vol. 3, Sorting and Searching*. Addison-Wesley Reading, 1973.

- [79] Ker-I Ko. *Complexity Theory of Real Functions*. Birkhauser, Boston, 1991.
- [80] E. Kondili, C.C. Pantelides, and R.W.H. Sargent. A general algorithm for scheduling batch operations. Technical report, Department of Chemical Engineering, Imperial College of Science and Technology, London, 1989.
- [81] Gary Koop, Eduardo Ley, Jacek Osiewalski, and Mark F.J. Steel. Bayesian analysis of long memory and persistence using arfima models. Technical report, Department of Economics, University of Toronto, May 1994.
- [82] H. Kuryla and J. Mockus. Bayesian heuristics "learning" in a flow-shop problem. *Informatica*, 1995. in print.
- [83] B. Lee and G.V. Reklaitis. Optimal scheduling of batch processes for heat integration, part i. *Computers & Chem. Eng.*, 1995. in print.
- [84] J. Lee. A novel design method for multilayer feedforward neural networks. *Neural Computation*, 6:885–901, 1994.
- [85] A.V. Levy, A. Montalvo, S. Gomez, and A. Calderon A. Topics in global optimization. In *Lecture Notes in Mathematics*, number 909, pages 18–33. Springer Verlag, Berlin, 1982.
- [86] W.K. Li and A. I. McLeod. Fractional time series modelling. *Biometrika*, 73:217–221, 1986.
- [87] S. Lin. Computer solutions of the travelling salesman problem. *Bell Systems Technical Journal*, 44:2245–2269, 1965.
- [88] J. Liu. On the existence of general multiple bilinear time series. *Journal of Time Series Analysis*, 10:341–355, 1989.
- [89] A.I. McLeod and K.V. Hippel. Preservation of the rescaled adjusted range. a reassessment of the hurst phenomenon. *Water Resource Research*, 14:491–508, 1978.
- [90] J. Michael. *The Computation of Fixed Points and Applications*. Springer-Verlag, Berlin, 1976.
- [91] V.S. Michanevich, A.M. Gupal, and V.I. Norkin. *Methods of Non-Convex Optimization*. Nauka, Moscow, Russia, 1968. In Russian.
- [92] D.L. Miller and J.F. Pekny. Exact solution of large asymmetric travelling salesman problems. *Science*, 251:754–761, 1991.

- [93] A. Mockus. *Predicting a Space-Time Process from Aggregate Data Exemplified by the Animation of Mumps Disease*. PhD thesis, Department of Statistics, Carnegie Mellon University, 1994.
- [94] A. Mockus, J. Mockus, and L. Mockus. Bayesian approach adapting stochastic and heuristic methods of global and discrete optimization. *INFORMATICA*, 5(1–2):123–166, 1994.
- [95] A. Mockus, J. Mockus, and L. Mockus. Bayesian approach to global and discrete optimization. Technical report, Optimization Department, Institute of Mathematics and Informatics, Vilnius, Lithuania, 1994.
- [96] A. Mockus and L. Mockus. Design of software for global optimization. *Informatika*, 1:71–88, 1990.
- [97] J. Mockus. Information based complexity and bayesian heuristics approach to global and discrete optimization. *Journal of Complexity*. submitted.
- [98] J. Mockus. *Multimodal Problems in Engineering Design*. Nauka, Moscow, 1967. in Russian.
- [99] J. Mockus. On bayesian methods of extremum search. *Automatics and Computer Technics*, 72:53–62, 1972. in Russian.
- [100] J. Mockus. *Bayesian approach to global optimization*. Kluwer Academic Publishers, Dordrecht-London-Boston, 1989.
- [101] J. Mockus. Application of global linear search in optimization of networks. In Ding Zhu Du and P.M. Pardalos, editors, *Network Optimization Problems*, pages 169–175. World Scientific Publishing Co., 1993.
- [102] J. Mockus and L. Mockus. Bayesian approach to global optimization and applications to multiobjective and constrained optimization. *Journal of Optimization Theory and Applications*, 70(1):155–171, July 1991.
- [103] J. Mockus and A. Soofi. The long-run economic relationships: An optimization approach to fractional integrated and bilinear time series. *INFORMATICA*, 6:61–70, 1995.
- [104] Jonas Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–366, June 1994.
- [105] L. Mockus and G.V. Reklaitis. Mathematical programming formulation for scheduling of batch operations using nonuniform time discretization. In *AIChE annual meeting*, number 235d, San-Francisco, California, 1994.

- [106] L. Mockus and G.V. Reklaitis. A new global optimization algorithm for batch process scheduling. In *Proceedings of State of the Art in Global Optimization: Computational Methods and Applications*, Princeton, NJ, 1995.
- [107] K.L. Musser, J.S. Dhingra, and G.L. Blankenship. Optimization based job shop scheduling. *IEEE Transactions on Automatic Control*, 38:808–813, 1993.
- [108] L. Nagel. Spice2: A computer program to simulate semiconductor circuits. Technical Report UCB ERL-M250, University of California, Berkeley, California, May 1975.
- [109] J. Neuman and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, 1953.
- [110] A. Newell. Heuristic programming: Ill-structured problems. In *Progress in Operations Research*, volume 3. John Wiley, 1969.
- [111] Nils Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, 1971.
- [112] G. Owen. *Game Theory*. W. B. Saunders, Philadelphia, PA, 1968.
- [113] E.W. Packel and H. Wozniakowski. Recent developments in information-based complexity. *Bulletin of the AMS*, 17:9–35, 1987.
- [114] P. Pardalos and Y. Siskos. Editorial, a historical perspective. In P.M. Pardalos and Y. Siskos, editors, *Advances in Multicriteria Analysis*, pages ix–xxv. Kluwer Academic Publishers, 1995.
- [115] P.M. Pardalos, K.A. Murthy K.A., and T.P. Harrison. A computational comparison of local search heuristics for solving quadratic assignment problems. *Informatica*, 4:172–187, 1993.
- [116] P.M. Pardalos, L. Pitsoulis, , T. Mavridou, and M.G.C. Resende. Parallel search for combinatorial optimization: Genetic algorithms, simulated annealing and GRASP. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems, Proceedings of the Second International Workshop –Irregular’95*, volume 980 of *Lecture Notes in Computer Science*, pages 317–331. Springer-Verlag, 1995.
- [117] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. FORTRAN subroutines for approximate solution of sparse quadratic assignment problems using GRASP. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1995.

- [118] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A parallel GRASP implementation for the quadratic assignment problem. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregularly Structured Problems – Irregular '94*, pages 111–130. Kluwer Academic Publishers, 1995.
- [119] P.M. Pardalos and J.B. Rosen. *Constrained global optimization: Algorithms and applications*. Springer-Verlag, Berlin, 1987.
- [120] A.N. Patel, R.S.H. Mah, and I.A. Karimi. Preliminary design of multi-product noncontinuous plants using simulated annealing. *Computers and Chemical Engineering*, 15:451–469, 1991.
- [121] G. Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890. English translation in *Selected Works of Guiseppe Peano*, 1973, Hubert C. Kennedy, Ed., University of Toronto Press.
- [122] J.F. Pekny and D.L. Miller. Exact solution of the no-wait flowshop scheduling problem with a comparison to heuristic methods. *Computers & Chem. Eng.*, 15:741–748, 1991.
- [123] S.A. Pijavskij. An algorithm for finding the absolute extremum of function. *Computational Mathematics and Mathematical Physics*, pages 57–67, 1972.
- [124] M.J.D. Powell. On the convergence rate of the variable metric algorithm. *Journal of Institute of Mathematics and Applications*, 7:21–36, 1971.
- [125] T. Subba Rao and M.M. Gabr. An introduction to bispectral analysis and bilinear time series models. In *Lecture Notes in Statistics*, number 24. Springer-Verlag, Berlin, 1984.
- [126] G. Rappl. *Konvergenzraten von Random-Search-Verfahren zur Globalen Optimierung*. PhD thesis, Hochschule der Bundeswehr, Munchen, Germany, 1994.
- [127] L.A. Rastrigin. *Statistical Methods of Search*. Nauka, Moscow, 1968. in Russian.
- [128] G.V. Reklaitis and L. Mockus. Mathematical programming formulation for scheduling of batch operations based on the nonuniform time discretization. Technical report, School of Chemical Engineering, Purdue University, West Lafayette, Indiana, 1994.
- [129] M.G.C. Resende. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1996. Personal Communication.

- [130] M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran subroutines for approximate solution of pardalos95apardalos95b, dense quadratic assignment problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, March 1996.
- [131] M.G.C. Resende and C.C. Ribeiro. A GRASP for graph planarization. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1995.
- [132] H. Robbins and S. Munroe. A stochastic approximation method. *Ann. Math. Statist.*, 22(1):400–407, 1951.
- [133] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representation by back-propagating errors. *Nature*, 323:533–536, 1986.
- [134] H.S. Ryoo and N.V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8:107–138, 1996.
- [135] N. V. Sahinidis and I. E. Grossmann. Reformulation of multiperiod MILP models for planning and scheduling of chemical processes. *Computers Chem. Engng.*, 15:255–272, 1991.
- [136] V. Saltenis. *Analysis of Structure of Multi-modal problems*. Mokslas, Vilnius, Lithuania, 1989. in Russian.
- [137] K. Schittkowski. Nlpql: A fortran subroutine solving constrained nonlinear programming problems. *Annals of Operations Research*, 5:485–500, 1985/86.
- [138] G.O. Shubert. A sequential method seeking the global maximum of function. *SIAM Journal on Numerical Analysis*, 9:379–388, 1972.
- [139] V. Vyšniauskas. Searching for minimum in neural networks. *Informatika*, 5:241–255, 1994.
- [140] V. Vyšniauskas. *Performance issues of neural networks*. PhD thesis, Institute of Mathematics and Informatics, Vilnius, Lithuania, 1996.
- [141] I.M. Sobolj. On a systematic search in a hypercube. *SIAM Journal on Numerical Analysis*, 16:790–793, 1967.
- [142] F. Sowel. Maximum likelihood estimation of stationary univariate fractionally integrated models. *Journal of Econometrics*, 53:165–188, 1992.
- [143] A.G. Sukharev. On optimal strategies of search of extremum. *Computational Mathematics and Mathematical Physics*, pages 910–924, 1971. in Russian.

- [144] H.A. Taha. *Integer Programming*. Academic Press, New York- San Francisco-London, 1975.
- [145] M. Tandon, P.T. Cummings, and M.D. La Van. Flowshop sequencing with non-permutation schedules. *Computers and Chemical Engineering*, 15:601–607, 1991.
- [146] G. Tesauro, Yu He, and Subutai Ahmad. Asymptotic convergence of backpropagation. *Neural Computation*, 1:382–391, 1989.
- [147] V. Tiesis. The method of variable metrics for local optimization of functions of many variables with rectangular constraints. In *Proceedings of the Conference on Computers*, pages 111–114, Kaunas, Lithuania, 1975. in Russian.
- [148] A. Torn and A. Zilinskas. *Global optimization*. Springer-Verlag, Berlin, 1989.
- [149] J.F. Traub, G.W. Wasilkowski, and H. Wozniakowski. *Information-Based Complexity*. Academic Press, New York, 1988.
- [150] J.F. Traub and H. Wozniakowski. Perspectives on information-based complexity theory. *Bulletin of the AMS*, 26:29–52, 1992.
- [151] V.G. Ustiuzhaninov. Random search in continuous problems of global optimization. In V.V. Fiodorov, editor, *Models and Methods of Global Optimization*, pages 37–45. Moscow, Russia, 1985. in Russian.
- [152] E. M. Vaisbord and D.B. Judin. Multi-extremal stochastic approximation. *Tekhnicheskaya Kibernetika*, (5):3–13, 1968. in Russian.
- [153] P. van der Smagt. Minimisation methods for training feed-forward networks. *Neural Networks*, 7(1):1–11, 1994.
- [154] P.J.M. VanLaarhoven, C.G.E. Boender, E.H.L. Aarts, and A.H.D. RinnooyKan. A bayesian approach to simulated annealing. *Probability in the Engineering and Information Sciences*, 3:453–475, 1989.
- [155] L.M. Vidigal, S.R. Nassif, and S.W. Director. Cinnamon: Coupled integration and nodal analysis of mos networks. In *23 -th ASM/IEEE Design Automation Conference*, July 1986.
- [156] M.C. Wellons and G.V. Reklaitis. Scheduling of multipurpose batch chemical plants, part 1: Formation of single product campaigns. *Ind. Eng. Chem. Res.*, 30:671–688, 1991.

- [157] M.C. Wellons and G.V. Reklaitis. Scheduling of multipurpose batch chemical plants, part 2: Multiple-product campaign formation and production planning. *Ind. Eng. Chem. Res.*, 30:688–705, 1991.
- [158] N. Weymaere and J. P. Martens. On the initialization and optimization of multilayer perceptrons. In *IEEE Transactions on Neural Networks*, volume 5, pages 738–751, 1994.
- [159] J. White and A. Sangiovanni-Vincentelli. *Relaxation Techniques for the Simulation of VLSI Circuits*. Kluwer Academic Publishers, Dordrecht-Boston-London, 1986.
- [160] Z. Xueya and R. W. H. Sargent. A new unified formulation for process scheduling. In *AIChE Annual Meeting*, St. Louis, MO, 1993.
- [161] Cheung Yin-Wong and K. Lai. Long-run purchasing power parity during the recent float. *Journal of International Economics*, 34:181–192, 1993.
- [162] Z.B. Zabinsky, R.L. Smith, and J.F. McDonald. Improving hit and run for global optimization. Technical report, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan, 1990.
- [163] M. G. Zentner, J. F. Pekny, D. L. Miller, and G. V. Reklaitis. RCSP++: A scheduling system for the chemical process industry. In *Proc. PSE'94*, pages 491–495, Kyongju, Korea, 1994.
- [164] A. Zilinskas. The method of one-dimensional multiextremal optimization. *Engineering Cybernetics (Tekhnicheskaya Kibernetika)*, 76(4):71–74, 1976.
- [165] A. Zilinskas. Optimization of one-dimensional multimodal functions, algorithm as-133. *Applied Statistics*, 23:367–375, 1978.
- [166] A. Zilinskas. Axiomatic approach to statistical models and their use in multi-modal optimization theory. *Mathematical Programming*, 22:104–116, 1982.
- [167] A. Zilinskas. *Global optimization: axiomatic of statistical models, algorithms and their applications*. Mokslas, Vilnius, Lithuania, 1986. in Russian.
- [168] A. Zilinskas and A. Zygliavskij. *Methods of Global Optimization*. Nauka, Moscow, Russia, 1992. in Russian.



## Index

- A posteriori distribution, 19
- A Posteriori Measure, 39
- A priori distribution, 3, 13, 19, 64
- Algorithm of Approximation, 32
- Algorithm of Randomized
  - Heuristics, 180
- Approximate Methods, 178
- Artificial neural network, 27, 164
  - feed-forward, 166, 170
- Asymptotic density, 20
- Asymptotics, 16
- Auxiliary function, 12
- Average Case Analysis, 19
- Batch Process Scheduling, 233
- Batch processes, 25
- Bayesian algorithms, 21
- Bayesian Approach, 3–4, 6, 11, 19–20, 23–24, 38
- Bayesian Heuristic approach, 3
- Bayesian Heuristic Approach, 4, 8–9, 11, 13, 24, 31, 40
- Bayesian heuristics, 25
- Bayesian Risk, 35
- Bilinear Time Series, 76
- Branch-and-Bound, 9, 22
- Clustering, 29
- Combinatorial linear programming, 229
- Common Blocks, 289
- Competitive Model, 119–120
- Complexity, 22
- Composite laminates, 27, 76
- Computed information, 32
- Conditional probabilities, 20
- Consistency conditions, 20
- Consistency conditions, 40
- Continuous function, 15
- Convergence of Bayesian Methods, 67
- Convergence of Deterministic Techniques, 48
- Convergence of Randomized Techniques, 50
- Convergence rate, 14
- Convergence, 15
- Convergence, 190
- Cost of Computing, 33
- Covariance, 20
- Creation of the Thumb-nails, 375
- Critical Moment, 141
- Cross-over and mutation, 225
- Cross-over and mutation, 267
- Cross-over rate optimization, 226
- Cross-over, 8
- Data Collection, 355
- Decision Algorithms, 36
- Delta Randomization, 184
- Description of Routines, 291
- Discrete Optimization, 43
- Discrete, 3
- Discriminant
  - linear, 156, 160, 166
  - linear

- parameterized, 160
- Display and Interactions, 372
- Display of Missing Values, 366
- Display of Vector Quantities, 366
- Dominant Analysis, 18
- Dynamic Display, 357
- Dynamic Graphics Equipment, 352
- Dynamic Graphics, 351
- Dynamic index data, 370
- Dynamic index objectives, 369
- Dynamic index, 27
- Dynamic Visualization Approach, 3, 25
- Effect of dynamic maps, 362
- Electric circuits, 27
- Epidemiological Model, 358
- Error of approximation, 32
- Error surface, 154, 161, 164, 174
- Errors, 37
- Estimation of the Deterministic Model, 91
- Event-Driven Techniques, 139
- Exchange Rate Forecasting, 83
- Expected deviation, 21
- Expert knowledge, 3, 23
- Exponential time, 21
- Exponential-Time Algorithms, 34
- Flow-shop problem, 27, 204, 345
- Flow-shop, 24
- Function
  - activation, 157, 160
  - discriminant, 156
  - multi-modal, 161
  - unimodal, 161
- Gap-Avoiding Randomization, 51
- Gaussian distribution, 159
- Gaussian, 20
- Genetic Algorithms, 8, 222–223
- Global Bayesian method, 292
- Global Line-Search, 131
- Global Methods, 331
- Global one-dimensional method, 304
- Global Optimization Software, 29
- Global optimization
  - stochastic, 6
- Global Stochastic Optimization, 192
- Global, 3
- Graphical Techniques, 351
- Gupta heuristics, 24, 205
- Heuristic Approach, 11
- Heuristic Priority Rules, 243
- Heuristic Programming, 6
- Heuristic, 3
- Heuristics, 22
- High-Voltage Networks, 134
- Highleyman's classes, 159
- Homogeneity, 20
- Ill-Defined Problems, 25
- Image Transformations, 364
- Immunological model, 26
- Information Operations, 32
- Information-Based Complexity, 31
- Initial temperature, 25
- Initialization, 154, 172
  - random, 172
- Integer Bilinear Programming, 227
- Interactions, 364
- Interactive Analysis, 25
- Interactive DOS software, 281
- Interactive Dynamic Maps, 364
- Interactive Icon Index, 369
- Interactive UNIX software, 282
- Isoline, 166
- Job-shop problem, 25, 215
- Knapsack problem, 27, 195, 345
- Knapsack, 24
- Kolmogorov's consistency
  - conditions, 20
- Lagrange multipliers, 29, 126
- Layout of the Thumb-nails, 374
- Learning Bayesian Heuristics, 210

- Learning Heuristics, 43
- Learning mode, 210
- Learning, 13, 154
  - back-propagation, 170
  - backpropagation, 154
  - conjugate gradient, 170
  - convergence, 154
  - Perceptron, 164
  - step, 170
- Likelihood Maximization, 85
- Linear Programming, 222
- Lipschitz constant, 15
- Lipschitz functions, 17
- List of Methods, 288
- Local Bayesian method, 320
- Local Methods, 332
- Local optimization, 16
- Long-Memory Processes, 83
- Lottery, 12
- Lower bounds, 9
- Main Program, 291
- Manufacturing Process, 354
- Markovian, 20
- Maximum Likelihood, 83
- Mechanical System of
  - Shock-Absorber, 74
- Menu System, 330
- Method of clustering, 298
- Method of extrapolation, 294
- Method of uniform search, 296
- Method of variable metrics, 310
- Minimax Approach, 4, 10, 14–15
- Minimax, 15, 17
- Minimization of Residuals, 86
- Minimizing Risk Functions, 66
- Minimum
  - global, 164
  - local, 164, 173
- Mixed Integer Line-Search, 136
- Monte Carlo, 16, 21
- Monte-Carlo Simulation, 123
- Multi-modal stochastic function, 13
- Multi-Modality Examples, 101
- Multiple Views, 365
- Mutation, 8
- Network optimization, 27
- Noise, 16
- Noisy functions, 29
- Non-differentiable functions, 29
- Non-linear differential equations,
  - 139
- Non-linear Regression, 75
- Non-smooth Randomization, 184
- Non-Uniform Discrete-Time
  - Algorithms, 236
- Non-Uniform Discrete-Time
  - Model, 28, 235
- Objective Function, 289
- Observation, 15
- Observations, 36
- One-Step Predictions, 343
- Optimization of networks, 131–132
- Ordering of the Thumb-nails, 374
- Organic Molecule Model, 79
- Original objective function, 12
- Orthogonal Polynomial
  - Randomization, 182
- Parallel Bayesian Algorithms, 45
- Parallel computing, 31
- Parallel Heuristics, 46
- Parallel Observations, 46
- Parallel Risk Optimization, 45
- Parameter grouping, 25, 27, 216
- Pareto Optimality, 19
- Pareto-Optimal Approach, 18
- Penalty function heuristics, 226
- Penalty function, 29, 241
- Perceptron, 155, 159
- Permutation Schedule, 204
- Plotting Sum of Squared
  - Residuals, 340
- Polymeric composition, 27
- Polynomial Randomization, 180
- Polynomial time algorithms, 24

- Portability routines, 323
- Portable Fortran Library, 280, 283
- Potentially optimal intervals, 18
- Power System, 134
- Predicting "Next-day" Rate, 114
- Predicting Exchange Rates, 338
- Radius of Information, 35
- Radius, 160–161
- Random Walk, 114
- Randomization Approach, 11
- Randomized decision, 22
- Randomized Heuristic Approach, 12
- Randomized Heuristics, 3, 179
- Randomized, 3, 22
- Randomizing Heuristics, 240
- Raw Data animation, 360
- Rectangular region, 29
- Regularization, 173
  - "weight decay", 173
- Risk function, 20
- Search for Equilibrium, 120
- Sections and Aggregation, 366
- Semi-global method, 306
- Semi-Monte Carlo Simulation, 97
- Sequential Decision Problem, 177
- Sequential Decisions, 38
- Sharp Polynomial Randomization, 181
- Shock-absorber, 26
- Simplified Decisions, 38
- Simulated Annealing, 25, 185, 253
- Smooth Animation, 360
- Smooth dynamic representation, 27
- Smoothing, 365
- Social Model, 125
- Software Examples, 337
- Software Initialization, 329
- Software Installation, 285, 328
- Software Requirements, 327
- Solution
  - multiple, 171
- Spherical angles, 160–161
- Squared Residuals Minimization, 87
- State-Task Networks, 235
- Stochastic approximation, 29
- Stochastic case, 25
- Stochastic Discrete Optimization, 193
- Stochastic flow-shop problem, 209
- Stochastic optimization, 22
- Taboo Search, 9
- Theory
  - optimization, 161
- Thermostable Polymeric Composition, 81
- Travelling salesman problem, 25, 27, 200
- Travelling salesman, 24
- Twin-Node Event-Driven Techniques, 146
- Twin-node technique, 139
- Uniform Discrete-Time Model, 28, 233, 235
- Uniform random search, 301
- Unit
  - hidden, 164, 170–171
- Updating A Priori Distributions, 65
- Users Reference, 327
- Video display, 360
- Visual indexing, 26–27, 376
- Weights
  - dynamics, 164, 170
  - initialization, 154
  - symmetries, 171
- Worst Case Analysis, 14
- Worst case, 22
- Yield of Differential Amplifiers, 72