WARSAW UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRONICS AND INFORMATION TECHNOLOGY

Mariusz Kamola, MSc

Algorithms for Optimisation Problems with Implicit and Feasibility Constraints

PhD Dissertation

Thesis Supervisor
Professor Krzysztof Malinowski, DSc

WARSZAWA 2004

Contents

Symbols and acronyms used

General notations

a	-	scalar
$\mathbf{a}$	-	vector or matrix[1]
a_i	-	i-th element of vector $\mathbf{a}$
$a_{i,j}$	-	element in i-th row and j-th column of matrix $\mathbf{a}$
$\{a_n\}$, $\{\mathbf{a}_n\}$	-	sequence of scalars (resp. vectors or matrices)
$\mathbf{a}^T$	-	transpose of $\mathbf{a}$
$a(\cdot)$	-	scalar function
$\mathbf{a}(\cdot)$	-	vector or matrix function
$a_i(\cdot)$	-	i-th scalar function constituting vector function $\mathbf{a}(\cdot)$
$a_{i,j}(\cdot)$	-	i-by-j-th function constituting matrix function $\mathbf{a}(\cdot)$
a_{L}, a_{U}	-	lower bound on a, upper bound on a
$\hat{a}$	-	some estimate of a
$\bar{\mathbf{a}}$	-	mean value of $\mathbf{a}$ elements
$\nabla a(\cdot)$	-	first order derivative of $a(\cdot)$, a vector function
$\nabla^2 a(\cdot)$	-	second order derivative of $a(\cdot)$, a matrix function
$\nabla_{\mathbf{b}} a(\mathbf{b}, \cdot)$	-	first order derivative of $a(\mathbf{b}, \cdot)$ w.r.t. $\mathbf{b}$
$\nabla^2_{\mathbf{bc}} a(\mathbf{b}, \mathbf{c}, \cdot)$	-	second order mixed derivative of $a(\mathbf{b}, \mathbf{c}, \cdot)$ w.r.t. $\mathbf{b}$ and $\mathbf{c}$
$\nabla \mathbf{a}(\mathbf{b})$	-	first order derivative of $\mathbf{a}(\mathbf{b})$, a $\dim \mathbf{a} \times \dim \mathbf{b}$ matrix function
A	-	set[2]
$\bar{\bar{A}}$	-	cardinality of set A
ξ, $\boldsymbol{\xi}$	-	random variable, multidimensional random variable
$\boldsymbol{\xi}_i$	-	i-th realisation of a multidimensional random variable $\boldsymbol{\xi}$

[1]Denoted always with a bold letter.

[2]Denoted always with a capital Latin letter.

Symbols invariant through this dissertation

$\mathbf{D}_\mathbf{x}$	-	set of $\mathbf{x}$'s feasible w.r.t. explicit constraints
$\mathbf{D}_\mathbf{y}$	-	set of feasible $\mathbf{y}$'s
$\mathbf{d}$	-	search or section direction
$f(\cdot)$	-	performance index (a.k.a. objective function)
$\mathbf{h}(\cdot)$	-	implicit functional relation between $\mathbf{x}$ and $\mathbf{y}$
$\mathbf{e}_i$	-	versor parallel to i-th axis
$\mathbf{I}$	-	identity matrix
k_S	-	routine termination parameter (see p. 87)
$\mathbf{q}$	-	vector of model internal variables
$\mathcal{R}$	-	set of real numbers
$\mathbf{s}(\cdot)$	-	explicit function mapping $\mathbf{x}$ into $\mathbf{y}$
t	-	time (continuous or discrete)
$\mathcal{U}$	-	random variable of uniform distribution
w	-	history window spanning w last evaluations of $f(\cdot)$
$\mathbf{x}$	-	vector of decision variables
$\mathbf{x}^\star$	-	problem solution
$\mathbf{y}$	-	vector of dependent variables
ϵ	-	accuracy (value used by calculations termination criterion)
ϵ_A	-	routine termination parameter (see p. 87)
ϵ_I	-	routine termination parameter (see p. 87)

Frequently used acronyms

CORBA	-	Common Object Request Broker Architecture
CRS	-	Controlled Random Search
DCOM	-	Distributed Component Object Model
EA	-	Evolutionary Algorithm
FDTD	-	Finite Difference Time Domain (simulation method)
FEM	-	Finite Element Method
ICCE	-	Institute of Control and Computation Engineering
IHE	-	Institute of Heat Engineering
IP	-	Internet Protocol
IPA	-	Infinitesimal Perturbation Analysis
IR	-	Institute of Radioelectronics

Frequently used acronyms (cont.)

LP	-	Linear Programming
LR	-	Likelihood Ratio
MPI	-	Message Passing Interface
NSP	-	Network Solution Provider
PM	-	Pricing Module
PVM	-	Parallel Virtual Machine
QoS	-	Quality of Service
QOSIPS	-	Quality of Service and Pricing Differentiation for IP Services (project)
QP	-	Quadratic Programming
QW-3D	-	*QuickWave-3D*
RMI	-	Remote Method Invocation
RPC	-	Remote Procedure Call
RSM	-	Response Surface Methodology
SA	-	Stochastic Approximation
SIMD	-	Single Instruction Multiple Data (architecture)
SLA	-	Service Level Agreement
SLP	-	Sequential Linear Programming
SLR	-	Sequential Linearisation with Relaxation
SOF	-	Simulation-Optimisation Framework
SQP	-	Sequential Quadratic Programming
WUT	-	Warsaw University of Technology

Acknowledgments

The author would like to express many thanks to his supervisor Professor Krzysztof Malinowski for his eager support and advice in all works that have led to this dissertation.

The author wishes to thank his colleagues, W. Bujalski, PhD (Institute of Heat Engineering) and P. Miazga, PhD (Institute of Radioelectronics), for the cooperation in joint projects on models presented here, and for their disinterested further help in numerical experiments.

The author thanks his family and colleagues for all indulgence, patience, interest and fruitful remarks.

The work reported in this dissertation on pp. 23–32, 85–86 was supported by PATIA programme.

The work reported in this dissertation on pp. 40–45, 113–116, 53–55, 123–125, 59–65, 142–156 was supported by the State Committee for Scientific Research under grant 7 T11A 022 20.

The work reported in this dissertation on pp. 47–53 has been carried out within IST-1999-20033 project supported by European Commission.

The work reported in this dissertation on pp. 65–80, 132–139 has been supported by the Rector of Warsaw University of Technology under grant 503G/1031/0010/003.

Fig. 2.9 has been published by courtesy of QWED, Ltd.

Chapter 1

Introduction

The subject of this dissertation are modelling and optimisation being used jointly, namely in optimisation problems where the performance index value is determined by the output from a model, as outlined in Fig. 1.1. Therefore, one will not be presented with procedures of model making or optimisation routine construction but rather with an interplay of the existing real-life models and state-of-the-art optimisation algorithms.

Looking back at last decades of rapid computing advancement, one may share the impression that optimisation did not benefit from it so much as modelling did. At least for the class of considered problems (i.e. with performance index based on model output) the increasing computational power was always consumed first by the model, and then optimisation routine took what was left over. Various expensive modelling approaches like Monte-Carlo simulation or Finite Element Method (FEM), let alone complex equations solvable only iteratively, just waited for available resources. This situation demanded optimisation routines serving such models to remain cost-efficient.

In cases when models were so simple that their output computation did not even deserve terms 'solving' or 'simulation', optimisation routines could evolve. Only then more complex optimisation problems could be formulated and solved, usually exhausting all the resources not used by the model. An appropriate example could be dynamic programming with its

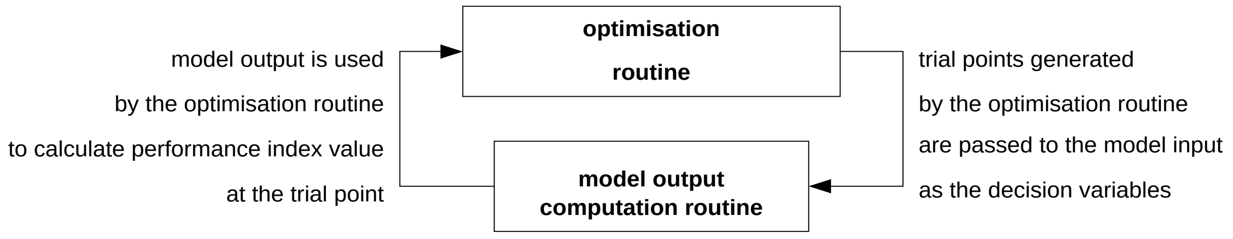

optimisation routine
model output computation routine
model output is used by the optimisation routine to calculate performance index value at the trial point
trial points generated by the optimisation routine are passed to the model input as the decision variables

Figure 1.1: Flow of data for optimisation problems with the performance index calculated from some model output.

Ford-Bellman algorithm, and with its dimensionality curse.

In presence of such apparent lack of computational power for complex models and advanced optimisation algorithms at the same time, one may ask how practical model-based optimisation problems are now solved. They are solved thanks to close integration and narrow specialisation of the modelling and optimisation algorithms. Close integration means that optimisation routine is created or modified to handle just the type of performance index that the model generates. On the other hand, the model provides the optimisation routine not only with its output, but with a series of internal variables, and possibly also with derivatives. Such simulation-optimisation programs are built to design optimal placement of objects, to determine optimal buffer size in networks etc.

There exist also general-purpose optimisation solvers with their own specific modelling languages. They owe their efficiency to limitation of the languages they provide. Plugging a third party model module into them is practically impossible. Again, limiting the number of language constructs made possible close integration of the model computation and optimisation routines at the cost of inapplicability for attacking nonstandard problems.

The author's main intention is to focus on those optimisation problems where one cannot require neither a particular open-form definition of the model nor any more data than the mere model output. Therefore, the initial question is:

1. Are there practical optimisation problems with performance index determined by a 'black-box' model?

2. How such problems are solved? How they are to be solved?

The former part of the above question does not have to wait long to be answered positively. In fact, the whole research was spawned by a project run jointly with Institute of Heat Engineering (IHE) at Warsaw University of Technology (WUT) with the purpose to find an optimal set-point for a power plant. The latter part of the question does not have a straightforward answer — this dissertation attempts to be the one.

Let us put formally the problem of optimisation with the performance index dependent on model output:

$$\min_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}) \ , \tag{1.1}$$

where $\mathbf{x}$ is the vector of decision variables, i.e. values decisive for the process of model output computation.[1] Those values are generated during the run of an optimisation routine inside

[1]The terms *model output computation*, or shorter *model computation*, and short *simulation* will be used here interchangeably. The term *modelling* refers rather to the process of model structure creation and model parameters tuning than to the actual process of model day-to-day use.

the optimisation module. In order to compute the value of $f(\cdot)$, they are passed to the model computation routine, inside the model module.[2] The model yields a vector $\mathbf{y}$ of dependent variables representing the model output, which is passed back to the optimisation routine and used to compute $f(\cdot)$. If the model guarantees $\mathbf{y}$ to be computed for every valid $\mathbf{x}$ then model computation can be written as computation of some function $\mathbf{s}$

$$\mathbf{y} = \mathbf{s}(\mathbf{x}) \ . \tag{1.2}$$

There are, however, models with the computation failing for, seemingly, valid value of $\mathbf{x}$. In such cases it is more appropriate to present simulation as a procedure solving equation set

$$\begin{cases} \mathbf{h}_1(\mathbf{x},\mathbf{y}) & = & 0 \\ & \vdots & \\ \mathbf{h}_N(\mathbf{x},\mathbf{y}) & = & 0 \end{cases} \ , \text{ in short,} \quad \mathbf{h}(\mathbf{x},\mathbf{y}) = \mathbf{0} \ , \tag{1.3}$$

with $\mathbf{h}(\mathbf{x},\mathbf{y})$ being a function taking value $\mathbf{0}$ if $\mathbf{y}$ is the output of successfully performed simulation with input value of $\mathbf{x}$.

Problem (1.1) is subject to two basic constraints

$$\text{a)} \ \mathbf{x} \in D_{\mathbf{x}} \ , \qquad \text{b)} \ \mathbf{y} \in D_{\mathbf{y}} \ , \tag{1.4}$$

the former one imposed on decision variables and the latter one on output variables, respectively. Mostly, the domains $D_{\mathbf{x}}$ for decisions and $D_{\mathbf{y}}$ for outputs take the form of simple hypercubes

$$\begin{aligned} &\text{a)} \ D_{\mathbf{x}} = \{\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} : x_{\mathrm{L},1} \le x_1 \le x_{\mathrm{U},1}, \ \ x_{\mathrm{L},2} \le x_2 \le x_{\mathrm{U},2}, \ \ \ldots\} \ , \\ &\text{b)} \ D_{\mathbf{y}} = \{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix} : y_{\mathrm{L},1} \le y_1 \le y_{\mathrm{U},1}, \ \ y_{\mathrm{L},2} \le y_2 \le y_{\mathrm{U},2}, \ \ \ldots\} \ , \end{aligned} \tag{1.5}$$

with $\mathbf{x}_{\mathrm{L}}$, $\mathbf{x}_{\mathrm{U}}$, $\mathbf{y}_{\mathrm{L}}$ and $\mathbf{y}_{\mathrm{U}}$ being vectors of lower and upper bounds for $\mathbf{x}$ and $\mathbf{y}$. Regardless of the internal workings of an optimisation routine, the only means of using the model is to feed it with $\mathbf{x}$ and wait for $\mathbf{y}$. Therefore, it is desirable to consider the practical effect (1.4b) has on the search domain. A relatively simple constraint (1.4b) together with (1.2) or (1.3) define additional, implicit constraints to (1.1). Moreover, problem may be additionally constrained

[2]The term *model module*, i.e. the name of a software module which carries out model output computation operations, will be used interchangeably with short *simulator*.

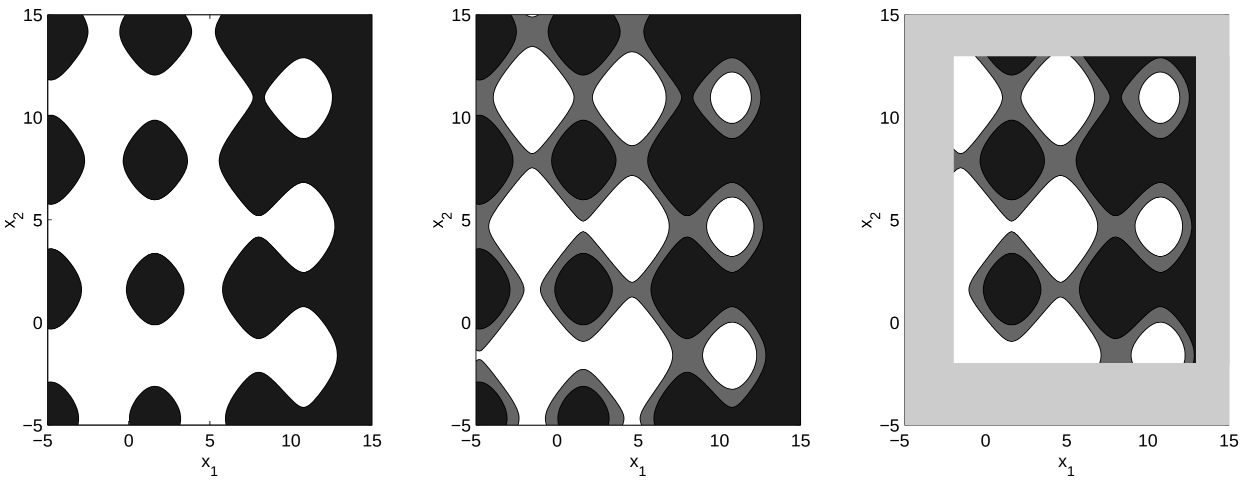


Figure 1.2: Left: The search domain with a region (indicated with dark gray) where the solution of (1.3) does not exist and feasibility constraints are violated. Middle: As on the preceding plot, and with a region (gray) where implicit constraints only are violated. Right: As on the preceding plot, and with a region (light gray) where the explicit constraints are violated. The region feasible for optimisation problem is therefore the white one.

by such $\mathbf{x} \in D_{\mathbf{x}}$ for that (1.3) cannot be solved. The optimisation problem (1.1, 1.3, 1.4) is thus subject to three types of constraints:

1. Explicit — defined by (1.4a);
2. Implicit — defined by (1.4b) together with (1.2): $\{\mathbf{x} : \mathbf{s}(\mathbf{x}) \in D_{\mathbf{y}}\}$ — or defined by (1.4b) together with (1.3): $\big\{\mathbf{x} : \exists_{\mathbf{y} \in D_{\mathbf{y}}} \mathbf{h}(\mathbf{x}, \mathbf{y}) = \mathbf{0}\big\}$;
3. Feasibility — defined solely by (1.3) — $\{\mathbf{x} : \exists_{\mathbf{y}} \mathbf{h}(\mathbf{x}, \mathbf{y}) = \mathbf{0}\}$.

Although implicit constraints define just a subset of feasibility constraints, it is important to distinguish them: when implicit constraints are violated, one still has some reliable value of $\mathbf{y}$; when the violation of feasibility happens, there exists no $\mathbf{y}$ whatsoever. Analytical calculation of sets of $\mathbf{x}$ satisfying implicit or feasibility constraints is impossible due to the advanced numerics underlying $\mathbf{s}(\cdot)$ and $\mathbf{h}(\cdot)$. There is no reason to expect those sets to be convex or compact. An exemplary shape those sets may take is shown in Fig. 1.2.

Feasibility constraints in wider context — coordination by the direct method

It should be emphasised that feasibility constraints, the most confusing and troublesome ones, are not specific only to simulation-optimisation problems. Let us present here another class of problems constrained in such way; those are optimisation problems with coordination by the direct method. The reader can find their detailed description in [40, pp. 36–37] or in [62].

Optimisation with coordination, or hierarchical optimisation, can be applied if a problem exhibits hierarchical structure. The hierarchy of the problem can be a natural consequence of hierarchical structure of the underlying model; it can also be handled 'technically' — as a set of problem properties, without investigating their reasons. In any case, the performance index to be handled by the direct method must be expressed as monotonic function $\Psi(\cdot)$ of performance sub-indices $f_1(\cdot), \ldots, f_N(\cdot)$

$$f(\mathbf{x}, \mathbf{y}) \overset{\text{df}}{=} \Psi\left(f_1(\mathbf{x}, \mathbf{y}_1), \ldots, f_N(\mathbf{x}, \mathbf{y}_N)\right) \quad , \quad \mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} \quad . \tag{1.6}$$

The problem of (1.6) minimisation can be then decomposed into a hierarchical decision problem with N local optimisation problems of finding

$$\mathbf{y}_i^\star = \arg \min_{\mathbf{y}_i \in D_{\mathbf{y}_i}(\mathbf{x})} f_i(\mathbf{x}, \mathbf{y}_i) \tag{1.7}$$

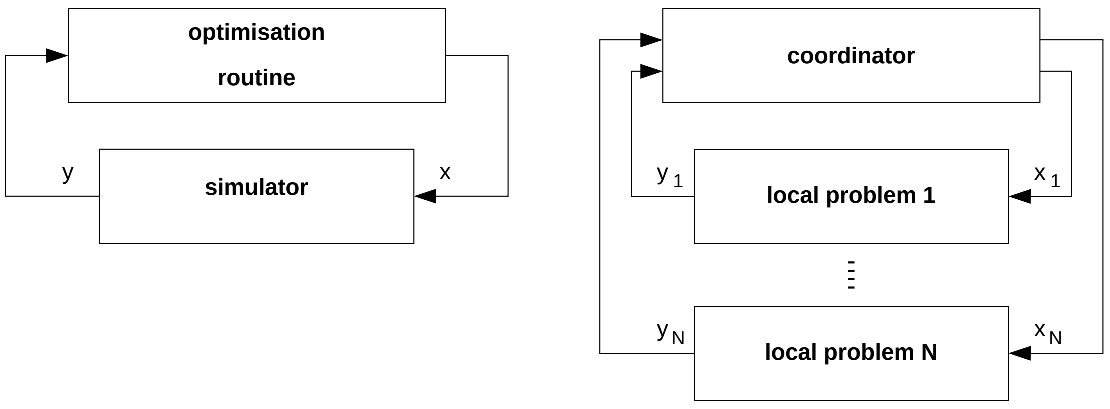
and the coordination problem[3]

$$\min_{\mathbf{x}} \Psi\left(f_1(\mathbf{x}, \mathbf{y}_1^\star), \ldots, f_N(\mathbf{x}, \mathbf{y}_N^\star)\right) \quad . \tag{1.8}$$

One of practical applications of such hierarchical optimisation structures is the control of a complex system, i.e. a system made of (possibly interconnected) blocks. Functions $f_i(\cdot)$ define goals for local controllers at block level, and $\Psi(\cdot)$ defines the global goal designed so as to not antagonise local controllers. The vector $\mathbf{x}$ of coordination variables specifies the desired values of block outputs as well as determines resources $D_{\mathbf{y}_i}$ for each controller. Under so defined conditions, local controllers do their best to provide control law optimising $f_i(\cdot)$'s.

There exist evident analogies between hierarchical optimisation and simulation-optimisation structures. They are presented in Fig. 1.3. The optimisation routine gets replaced by the coordinator (doing the same thing), and the simulator gets replaced by several local optimisation routines. However, the trouble with constraints remains the same. Particularly, feasibility constraints may equally well jeopardise operation of hierarchical optimisation. Note that, for certain $\mathbf{x}$'s some of $D_{\mathbf{y}_i}$'s may become empty, resulting in infeasible optimisation problem (1.7) definition and in no $\mathbf{y}_i^\star$ produced altogether — which is essentially the same situation as in

[3]Only the layout of hierarchical optimisation is given here to make the reader aware of the two problems similarities, and therefore to justify the need to investigate such problems. For more on hierarchical structures, see [40].



Simulation-optimisation		Hierarchical optimisation
optimisation — (1.1)	$\sim$	coordination — (1.8)
simulation — (1.2) or (1.3)	$\sim$	local problem — (1.7)

Figure 1.3: Data flow in simulation-optimisation (left) and in hierarchical optimisation (right) problems. The main differences consists in the number of modules serving performance index calculations (one resp. many) and in those modules nature (simulation resp. optimisation). Below are given the numbers of equations describing equivalent modules in both cases.

simulation-optimisation case. It is impossible to prevent this by requirement for the coordinator to take into account the effects all $\mathbf{x}$'s have on $D_{\mathbf{y}_i}$'s. One of the reasons is such effect is costly to compute, and comes into light only as the result of emitted coordination signals. Neither it is possible to assume that a good measure of the degree of $D_{\mathbf{y}_i}$ violation is available: consider system control example; a resource insufficient to carry out control may even result in loss of stability.[4]

In the course of his doctorate studies, the author was trying to answer the question stated above — are optimisation problems with performance index determined by some impenetrable model, and with the three types of constraints, really existent and practically solvable? In general, the answers received from the literature were negative. Many authors made it the precondition of problem solving that insight as deep as possible must be made into the nature and the specifics of a system being modelled. Such insight might then give hints for construction of an optimisation method trimmed exactly to the problem.

[4]Inconveniences caused by feasibility constraints in the direct method can be avoided by using coordination by the price method where Lagrange function is formulated as the objective, and the signals emitted by the coordinator to local problem solvers are the Lagrange multipliers. However, this is done at the cost of more restrictive assumptions for the performance index: $f(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{N} f_i(\mathbf{x},\mathbf{y}_i)$. Moreover, this approach does not guarantee solution optimality if duality gap occurs [62].

As regards possibility of solving such problems without interfering into the model module, practice slowly yielded positive response for particular encountered problems. The experience acquired while dealing with those problems may become the base for a methodology. The author distinguished common features of problems and common actions taken to solve them. Backed by a number of practical applications, he dares to generalise and to summarise the results of his work in theses that follow in Section 1.1.

1.1 Theses

1. There exist practical optimisation problems where the performance index value is based on output values obtained from some autonomous model fed with the decision variables. The coupling between simulation and optimisation module is weak, i.e. no more than decisions and model output are exchanged. Moreover, in general no knowledge is available about the nature of the system defined by the model. Formally, the optimisation problem is defined by (1.1), the action of model output computation is defined by (1.3), and the constraints are defined by (1.4), where $D_{\mathbf{x}} \subset \mathcal{R}^{\dim \mathbf{x}}$ and $D_{\mathbf{y}} \subset \mathcal{R}^{\dim \mathbf{y}}$. Practically, such problems can present difficulties because of

    - the fact that constraints (1.4b) on $\mathbf{y}$ additionally and implicitly constrain the domain in unpredictable way due to complicated nature of (1.2) or (1.3);
    - the fact that the process of computing the model output may be vulnerable to failures resulting from the modeller defects, but also representing forbidden states of the object being modelled.

2. Problems defined in Thesis 1 are in position to be solved by appropriate numerical algorithms run by a computer. Such algorithms have hybrid structure, i.e. they consist of a couple of relatively simple routines that are activated in the course of optimisation whenever their characteristics match best a particular stage of optimisation, and when they promise best results. Switching is performed between those component algorithms, i.e. at some point another routine takes over the optimisation task, utilising results obtained by its predecessor. To accomplish this effectively, the following actions ought to be carried out in algorithm design phase for every optimisation problem:

    (a) Appropriate component routines have to be chosen out of candidate routine set. The selection should be based on any initial knowledge of the problem, routines efficacy and availability of distributed computing environment.

   (b) Modifications of the selected component routines must be made if it turns out that, despite the selection based on all *a priori* information, the routines are not able to cope with the problem fully. In most cases those modifications concern building in the support for nonconvex optimisation domain.

   (c) Reasonable criteria of switching between component routines must be established. Those criteria base on the idea of progress understood not solely as the improvement of performance index but they may take into account such parameters as dispersion of trial points pool maintained by an algorithm etc.

   (d) If available, one should make use of parallel or distributed computing environment to accelerate, coarse by their nature, calculations.

3. It is possible to classify optimisation routines and problems by possession of attributes from a fixed set. Using such classification, one can try to perform automated selection of component algorithms that suit best the problem. Therefore, one can, with considerable confidence, extract a set of candidate component routines that will form a base for solving most problems. Furthermore, it is possible to hand the burden of quantitative definition of terms 'progress' and 'effort' over to the user, and to come up with a general-purpose environment for simulation-based optimisation with universal switching rules.

1.2 Comments on theses

The collective comment on all the theses is that surely they are not mathematically provable. What one can do is, for thesis with 'there exist' quantifier (Thesis 1), to indicate concrete real-life examples of optimisation problems and, for theses claiming 'for all' quantifier (Theses 2 and 3), to provide a number of convincing examples.

In Thesis 1, the question of constraints is particularly due an early comment. Implicitly constraining the search domain by introducing relatively simple bounds for $\mathbf{y}$ is a well-known fact. However, the critics can be put on the other source of constraining the domain, namely the failure of model output computation. Although, from the mathematical point of view, there is nothing unusual in violation of (1.3), a common sense and good engineering habits prompt to protest against flawed software incapable of passing a word of warning to the outer world.

In most cases this happens just because of the engineering approach that underlay the model construction. There may exist senseless $\mathbf{x}$ that makes the model stray away, hang or crash. Such $\mathbf{x}$ would never have passed the mind of an engineer skilled in the operation of modelled object. However, such $\mathbf{x}$ is a trial point generated by a general-purpose optimisation routine. Often, detecting the set of correct $\mathbf{x}$'s that do not result in simulation failures would cost more than the optimisation process itself. Similarly, tracing all the posts in model code with aim to warn of the danger is costly, if possible at all. Chapter 2 gives further explanations and discussion on this topic.

Thesis 2 proposes a methodology of attacking problems defined in Thesis 1. All the suggested activities 2a–2d lead to the construction of an algorithm capable of solving a given problem. It must be stressed that, unlike suggested in literature, the resulting approach does not attempt digging every possible bit of information from the model (or from its creators). The suggested methodology remains low-cost in application to a problem. (Obviously, it turns out to be to some extent high-cost at operation.) Neither the proposed approach aims at creation of a universal solver. The methodology takes the middle way: it collects pieces of others' algorithms only to put them together in a hybrid tool roughly adapted to a particular problem. Such approach is justified if off-the-shelf specialised simulators are to be embedded in some optimisation scheme. For more on this topic, see Section 2.2.

At the very start of component algorithm selection, in Thesis 2a appears the notion of 'candidate routine set'. The survey of algorithms recognised for simulation-based optimisation takes place in Chapter 3 and in Appendix A. The author's effort was put to cover all the fields where 'simulation' and 'optimisation' keywords appear together, to give an outlook as to where most burden and workload is currently placed.

Thesis 2b handles the case when complications appear during the operation of the constructed algorithm. They are inevitable, at least thanks to nonconvex or even not connected search domain that could be created by constraints on $\mathbf{y}$. Of the initial set of candidate algorithms, possibly only random search could support such constraints in its original shape. Rather than limiting oneself to random search only, it is worth to take more optimistic course, and to accept little more demanding but still simple routines in good belief that modifications will not be necessary. Should the problems occur, only then proper workarounds and patches (like appropriate repair algorithms) have to be developed. Chapter 4 presents how such approach worked in practice.

The essence of Thesis 2c is 'if the currently used component routine has run out of steam, start another one from the point where the old one reached'. What gives most trouble here is to determine whether the current algorithm has lost its efficiency permanently, or is just in the exploration phase? Exemplary switching criteria are given through Chapter 4, Section 5.1, and Section 6.2.

Thesis 2d underlines benefits coming from running the optimisation in parallel. Costly simulation underlying evaluations of $f(\cdot)$ makes the computation coarse-grained, and therefore adequate for parallelisation. In the considered situation, with the simulator being an opaque piece of software, one cannot actually speed up a single simulation. The only way to accomplish parallelism is through multiple simulations run at the same time to compute $f(\cdot)$ at various trial points — i.e. through parallelisation of the optimisation routine. A number of such routines are available and, since in any case one is to make effort of creating an interface between optimisation routine and simulator, it is worth to take pains to interface parallel routines. This thread will appear through Appendix B.

Thesis 3 presents several generalising conclusions the author tries to draw out of the practical examples whose nature was described in Chapter 2 and their solution in Chapter 4 and Chapter 5. Systematisation of optimisation problems is carried out in Section 6.2 that leads to a set of basic properties a simulation-optimisation problem may, or may not, present. Attributing a problem with those properties, as well as defining the measure of progress and computation effort (i.e. all the things that caused most troubles in Chapter 4 and in Section 5.1) are ceded to the user. Having accepted this, the user is presented with a prototype of simulation-optimisation environment that, hopefully, will do most of the hand work (appropriate algorithm selection and switching) described in Chapter 4 and Chapter 5.

This dissertation is arranged as follows. Presentation of practical simulation-optimisation problems with feasibility or implicit constraints is given in Chapter 2. The overview of algorithms widely recognised by other authors for simulation and optimisation problems is presented in Chapter 3 and Appendix A. Solving of the practical problems is covered in Chapter 4 and Chapter 5. Propositions of problems systematisation, and of a framework for automatic classification and application of selected algorithms towards a problem, are given, together with the concluding remarks, in Chapter 6.

Chapter 2

Practical problems with feasibility or implicit constraints

All reasoning and experimentation reported in this dissertation is done in context of three practical problems. Those problems description, given in subsequent sections, focuses strongly on difficulties they present to optimisation. In order to provide appropriately detailed problem description, some results from optimisation phase have to be revealed here beforehand. Therefore, the following material is partially the author's original contribution. For various reasons, diversified attention and effort was given to find solutions of the considered problems. The most exploited one is the power plant problem whose solving occupies whole Chapter 4. Solution procedures for the two other problems, as no so prominent, are enclosed collectively in Chapter 5.

2.1 Power plant model

In this section, the practical problem of industrial power plant set-point optimisation is presented. This problem was reported and solved with the support of IHE. That institute provided the author with appropriate modelling software and professional assistance.

The problem of power systems modelling has been for long present in the community of power engineers, and reflected in international as well as in Polish, literature [89]. It is a complex task indeed since the number of elements in the modelled object may reach thousands,[1] they are of various nature and, usually, of considerable complexity. Two main approaches exist to the model construction. The first is to establish a polynomial model of appropriate order for an element being modelled, and to identify its parameters given sufficient

[1]The works towards effective modelling of such complex systems are currently in progress in IHE.

amount of measurement data. The second approach makes use of the knowledge of element specifics, expressed in the form of a number of equations. Those equations may span various branches of science (e.g. thermodynamics, hydrodynamics, mechanics etc.) and attempt to model element behaviour as a function of some parameters known already at the stage of element design and construction.

The advantage of polynomial models is their simplicity and easiness of tuning — provided that the data for tuning are available. Unfortunately, while some of those data are available in great amount, some other are dramatically few or they are very difficult to be obtained from the living system.[2] An example can be measuring of temperature and pressure through a group of turbine stages; it is practically impossible due to extreme conditions there. Also the accuracy of such measurement is insufficient.

The advantage of a so-called physical model is absence of the requirement for extensive data collection to assure proper tuning. However, in order to model the system accurately, one has to develop sometimes highly complex mathematical formulae. This approach is prone to huge errors if the knowledge underlying formula construction is inadequate. Moreover, complexity of model solving, apart from the computational burden, has serious consequences in model vulnerability. Those consequences will be presented in throughout the section.

Physical models solving

The general structure of a coal industrial power plant does not differ from that of other power plants. It comprises of boiler, turbine and regeneration blocks [75, p. 111 and on], with water circulating through them in cycles. Attempts to construct numerical models of such systems or their parts are reflected in Polish scientific press for more than three decades. The particular thread leading to the model developed by IHE can be found in [58]; the authors report a mathematical model of a turboset (i.e. turbine, generator and heat exchangers). A physical model like the one reported is usually formed by a system of equations

$$\left\{\begin{array}{rcl} r_1(\mathbf{q}) & = & 0 \\ & \vdots & \\ r_N(\mathbf{q}) & = & 0 \end{array}\right. \tag{2.1}$$

representing physical interactions taking place in the system. System (2.1) is in fact system (1.3), but expressed in more detailed way; here, $r_1(\cdot), \ldots, r_N(\cdot)$ denote functions of model

[2]A similar case with imbalanced amount of data for proper model tuning is Internet Protocol (IP) services market model, described in Section 2.3. For the discussion on that model construction and tuning, see [9].

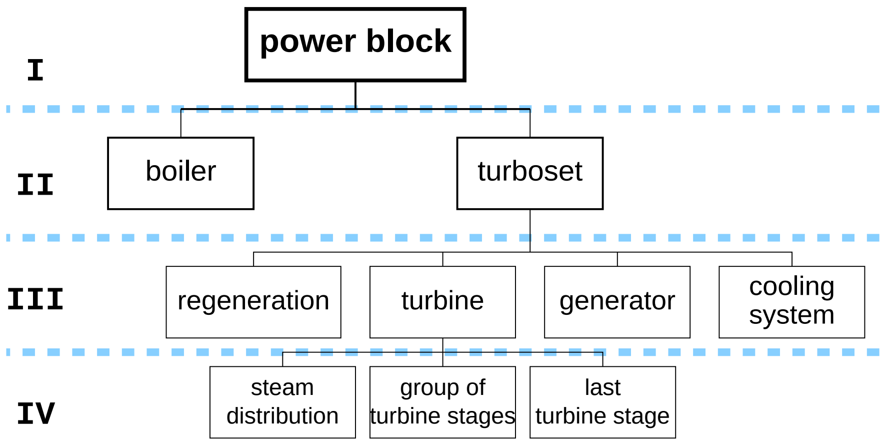


Figure 2.1: Levels of generality that can be determined in power system modelling — from most general (I) to most detailed (IV).

internal variables vector $\mathbf{q}$. (The vector $\mathbf{q}$ contains, in particular, also the model input $\mathbf{x}$ and the model output $\mathbf{y}$.)

The equations in (2.1) can be classified into three categories, depending on the modelled phenomena. They are: balances, states of turboset elements and properties of the working medium [74]. Equations of the first kind are always linear and are precisely determined by the graph of connections between the turboset elements. Complexity of the other equations depends on the type of elements being modelled; they are mostly nonlinear and, especially for second kind elements, troublesome in formulation.

System (2.1) has a number of degrees of freedom, and formally it is up to the user which elements of $\mathbf{q}$ will be adopted as the model input variables. However, careful selection of internal variables that will be used as the model inputs greatly reduces the solving time and the quality of numerical solution, if it is not conditioning the simulation success altogether. Two methods are proposed to solve system (2.1); the first is supported by the authors cited above and consists in decomposition of it into parts that could be solved sequentially or iteratively. Groups of equations to be solved sequentially form autonomous subsystems of (2.1), and are solved by consecutively substituting unknown variables. Groups of equations to be solved iteratively, are solved by an algorithm computing subsequent solution approximations until the desired accuracy is reached. A single iteration n of such algorithm may be written as follows

$$\tilde{\mathbf{q}}_{n+1} = \tilde{\mathbf{q}}_n + \tilde{\mathbf{r}}\left(\tilde{\mathbf{q}}_n\right) \quad , \tag{2.2}$$

where $\tilde{\mathbf{q}}_n$ is a subvector of $\mathbf{q}$ representing internal variables subject to iterative solving. Function $\tilde{\mathbf{r}}(\cdot)$ represents some procedure calculating corrections to be made to $\tilde{\mathbf{q}}$ in order to balance the relevant part of (2.1). The algorithm (2.2) requires some initial $\tilde{\mathbf{q}}_0$ to be specified. So, in this method some equations are solved once, some others are solved repeatedly, until the desired accuracy is reached. In complex power systems one can figure out a hierarchy of components; this is reflected by proposition [74] of hierarchy in the models, as in Fig. 2.1. In terms of models, complexity means the number of equations of the order of hundreds or thousands, and hierarchy means the solving procedure containing many nested loops.

Another method of solving (2.1) was proposed in [114]. It tends into the opposite direction; the author aggregates the model where possible to end up with a system of less than twenty equations. Only then he is able to express the task of finding the solution as an optimisation problem

$$\min_{\mathbf{x}} \sum_{i=1}^{N} r_i^2(\mathbf{q}) \quad , \tag{2.3}$$

with $\mathbf{x}$ being decision variable subvector of $\mathbf{q}$. Adapted quasi-Newton optimisation method is used to solve (2.3). Here also, setting appropriate initial values of some internal variables conditions prompt and successful optimisation.

IHE modelling software

The modelling software adopted and used by the author was created in IHE with an intention to assist system engineers and operators in predicting the effect that changes of selected parameters would have on the whole system operation in the steady state, i.e. when all the transient effects have ceased. Components and behaviour of IHE software, prevalently demanding user interaction, allow to qualify it as a complex support tool for manual working point selection. The software contains also a graphic editor allowing to create from the scratch the graph representing a power system, using elements of pre-defined properties from the library. Properties of those elements can next be adjusted. A model created this way is fed via file interface into the actual model solver.

Element types, connections and properties determine set (2.1) unambiguously. To solve it, the sequential-iterative method is applied. Since the 'solving path', i.e. the order in which the equations are grouped and solved can have tremendous impact on efficiency, pre-solving routine is written that, with operator assistance, determines the most efficient grouping of equations for a given set of internal variables selected for the model input $\mathbf{x}$.

In the course of model solving, the algorithm (2.2) is executed in the parts requiring iterations. Let us consider an example with $\mathbf{q}^T = [q_1 \; q_2 \; q_3]$ and with q_3 as the decision variable, $\mathbf{x}^T = [q_3]$. Let $r_1(\cdot) = r_1(q_2, q_3)$ and $r_2(\cdot) = r_2(q_1, q_2)$ be some linear and nonlinear functions, respectively. Then the solving procedure would require user to specify the initial value of one internal variable that will be computed iteratively, q_1 in this case, in order to start computations. Next, the solving algorithm would proceed as follows:

> STEP 0: Solve $r_1(q_2, q_3) = 0$ for q_2 once;
>
> STEP 1: Calculate $r_2(q_1, q_2)$; if $|\frac{r_2(q_1,q_2)}{q_1}| < \epsilon$ then stop, else set $q_1 := q_1 + \tilde{r}(q_1, q_2)$ and go to STEP 1.

The construction of $\tilde{r}(\cdot)$ can present itself as a difficult task since $\tilde{r}(\cdot)$ is to give the direction of adjustments necessary for q_1 in order to balance $r(\cdot)$. In general, $\tilde{r}(\cdot)$ has to do with the gradient $\nabla_{q_1} r_2(\cdot)$, which is not always existent or computable. It is the many-year experience that resulted in working $\tilde{\mathbf{r}}(\cdot)$'s in IHE simulator.

The approach to the construction of functions $r_i(\cdot)$ which are atoms for modelling elements behaviour, centered around the idea of nominal working point. Such a working point exists for many power system elements. The nominal working conditions have therefore become the starting point for calculation of *changes* of working conditions as functions of setting of some parameters (or model inputs) off their nominal values. The immediate consequence of such incremental, or small-signal modelling approach is that the model is valid only in a certain neighbourhood of the nominal working point. Yet, considering complexity of some elements, one cannot expect a nice locally linear model to emerge as a compensation. Taking group of turbine stages as an example, the modelling formulas for the flow below and above the nominal value are of completely different type. The implications are therefore threefold:

- For certain values of parameters far enough from the nominal ones the model ceases to be accurate;
- For values of parameters still more away from the nominal the mathematical formulas may not be computable;
- Even for parameters in the valid range, the model can be highly nonlinear.

Of course, in the case of the simulation software being operated by a skilled, experienced and trained person, none of the above disadvantages has to be taken into consideration. However, should the simulator be used by a general purpose optimisation module, all of the above implications are of importance.

Minimal running cost problem

In a coal power plant the energy from coal combustion in a boiler is received by water, which is the working medium. The water, in the form of steam, goes through other devices of the plant, giving away its energy and changing its parameters. The most complex receivers of steam are turbines of turbosets. In a turbine the steam gradually decompresses and cools down on a series of propellers in the three turbine parts (high-, medium- and low-pressure), spinning them. Usually, it is possible to let out some steam of various parameters at several extractions along its passage through the turbine. The output from a turbine are mechanical power, converted subsequently by the generator into electricity, and heat energy carried by steam flows of different parameters. The steam, to be warmed again, has to be regenerated, i.e. condensed in condensers, cooled down in heat exchangers, deareated and pumped, under high pressure, again into the boiler. Big systems may consist of many such power blocks (boiler–turbine–regenerator), interconnected through collectors to ease working medium distribution in order to react to e.g. changing power demand or to failures.

Apart from decompression, hot steam can also be used for warming. Its most popular application is to deliver heat to residential areas. Last but not least, it is used to warm up materials during production cycle in factories. In fact, this is the purpose industrial power plants are built for. This is also the case for the factory of our interest. The main goal of its power plant is to warm up materials for chemical reactions up to exact temperatures. The diagram of the considered plant is given in Fig. 2.2. The factory needs three types of steam of exact pressure, temperature and flow, and one of partially adjustable flow. Steam of desired parameters comes from taking and mixing steam from various points in the installation. It must be emphasised that the production of this 'industrial steam' is the absolute priority of the plant; electricity is just a side product. It is worth noticing that some of the parameters of the 'industrial steam', formally belonging to model input variables, remain fixed during a single optimisation run, and therefore are treated rather as model parameters than as the decision variables. They remain outside the vector $\mathbf{x}$ as well as a number of other input variables whose values are pre-set by a competent operator and are constant.

The factory management is interested in minimising running costs of the power plant by setting its steady-state working point appropriately. The running costs are defined as simple balance of cost of coal consumption, cost of pumps operation and the gains from selling the electric power;

$$f(\mathbf{x},\mathbf{y}) = c_{\mathrm{C}} \sum_{i=1}^{\dim \mathbf{y}_{\mathrm{C}}} y_{\mathrm{C},i} + c_{\mathrm{E}} \left(\sum_{i=1}^{\dim \mathbf{y}_{\mathrm{P}}} y_{\mathrm{P},i} - \sum_{i=1}^{\dim \mathbf{y}_{\mathrm{E}}} y_{\mathrm{E},i} \right) \quad , \tag{2.4}$$

where $\mathbf{y}_{\mathrm{C}}^T = [y_{\mathrm{C},1}\ y_{\mathrm{C},2}\ \dots]$ is the vector of coal flows (expressed in kg/sec), $\mathbf{y}_{\mathrm{P}}^T = [y_{\mathrm{P},1}\ y_{\mathrm{P},2}\ \dots]$ is the vector of power consumption by plant pumps (expressed in kilowatts), $\mathbf{y}_{\mathrm{E}}^T = [y_{\mathrm{E},1}\ y_{\mathrm{E},2}\ \dots]$ is the vector of power levels in generators, and c_{C} and c_{E} are coal and electricity prices, respectively. The need to adjust the plant working point such that (2.4) is minimised emerges several times a day, mainly because of the changing steam demands following the production schedule, usually known in advance. The performance index $f(\cdot)$ may also be affected by

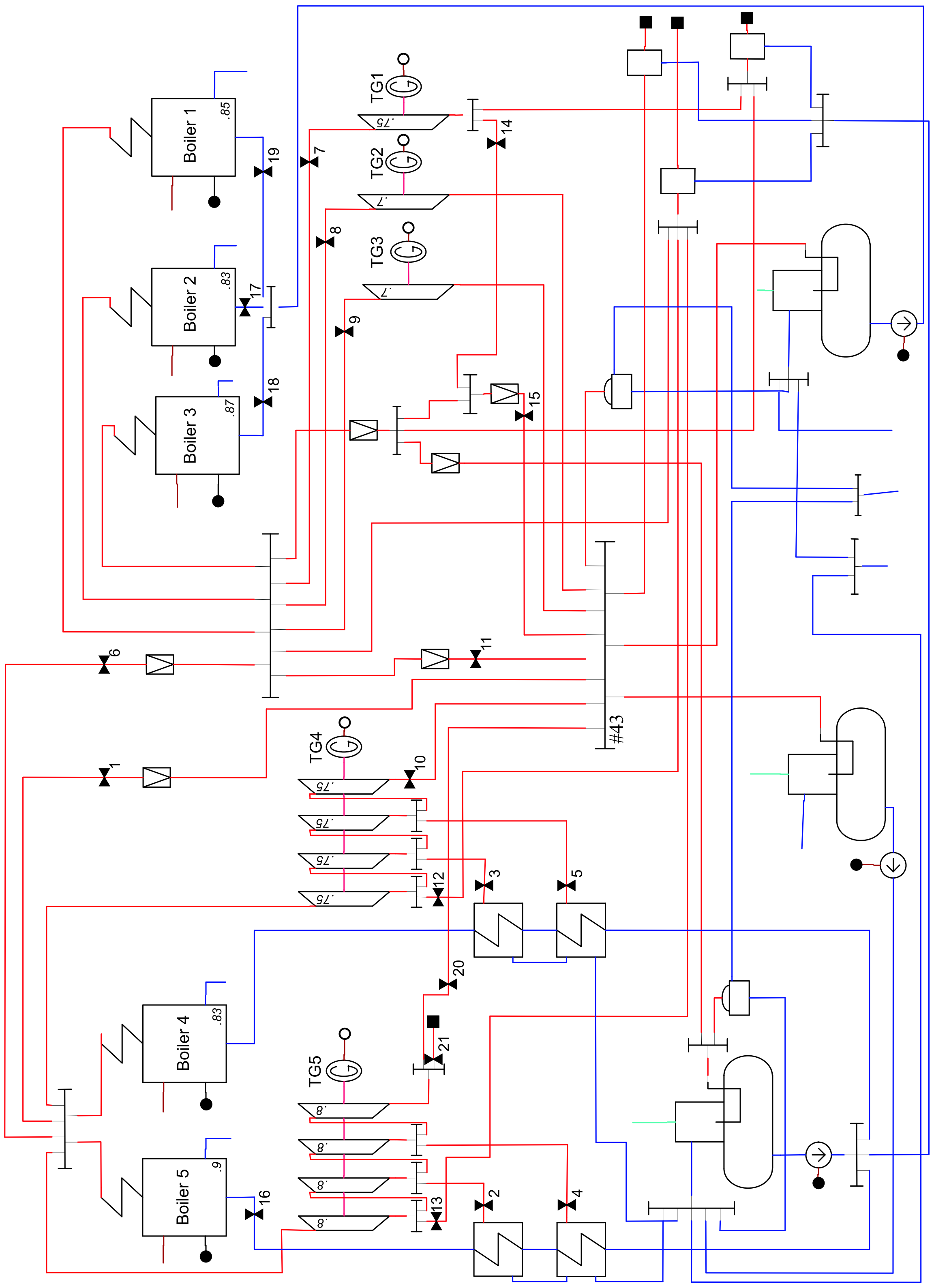


Figure 2.2: The diagram of industrial power plant at 'Janikosoda' chemical works in Janikowo as generated by IHE modelling tool. (Efficiencies of selected devices are given in italics.)
⧗ flow that is a decision variable (with the decision variable index given beside). ■ industrial steam outlet. ● flow or power level contributing to positive $f(\cdot)$ component in (2.4). ○ power level contributing to negative $f(\cdot)$ component.

changing electricity price c_{E} — actually this is the case in Poland due to the recent regulations in law — but those changes are currently not big.

As it was mentioned above, from the formal point of view (if the model has unambiguous solution $\mathbf{y}$ for an $\mathbf{x}$), the decision which internal variables are to be adopted as the model input is irrelevant. In practice, to secure efficient model computation, they are selected by a skilled engineer. Those selected for the considered optimisation problem are indicated in Fig. 2.2 and listed in Table 2.1. The choice may seem uncomfortable — one might want to include $\mathbf{y}_{\mathrm{C}}$, $\mathbf{y}_{\mathrm{P}}$ and $\mathbf{y}_{\mathrm{E}}$ in $\mathbf{x}$ first — but that is the best one can do to simplify calculations.

The main purpose of constraints formulation is to curb the range of decision variable values in attempt to approximate roughly the set a trained human operator would consider the reasonable model inputs. Moreover, upper and lower bounds on $\mathbf{x}$ bracket the nominal values of the corresponding internal model variables, and so reflect also the nature of modelling. These explicit constraints are summarised in Table 2.1.

All the internal model variables are available to the outer world as the model output, $\mathbf{y} = \mathbf{q}$. All they are subject to inequality constraints of the same type, $y_i \geq 0$, $i = 1, \ldots \dim \mathbf{y}$. This prevents the model from entering evidently invalid area. As it will turn out, such constraint specification, oriented to secure proper working conditions of every single element modelled, does not take into account interactions of elements and therefore does not prevent situations when modelling formulae domain is violated and no output is produced.

A special value for a decision variable determining flow that in certain cases lies outside $D_{\mathbf{x}}$, is zero. In reality zero flows are absolutely correct and denote the fact that a part of the system is off. Deciding which part of the system has to be working for satisfying the current demands is an important class of problems power industry, called Economic Load Dispatch (ELD) problems. In terms of problem formulation it leads to mixed programming problem (i.e. optimisation with decision variables taking values both from continuous and discrete sets). Such problems are not considered in this dissertation.[3]

Power plant model implementation

IHE model of the considered industrial power plant is made of 67 elements and 539 internal model variables representing flows, entalpies, pressures, temperatures, working parameters, efficiencies and power levels. (Part of those are slack variables, i.e. they serve to express

[3]Another interesting class of problems directly related to the considered one is not only *what* should the new optimal working point be, but *how* to switch to it. Specifically, what should be the trajectory in $D_{\mathbf{x}}$ that carries the system slowly (i.e. discarding the transient effects) to the new point, possibly skirting regions where constraints are active. This class of problems will neither be considered here.

Decision variable name	Internal model variable name	Lower bound	Upper bound	Description
x_1	q_{18}	0	5	steam flow through reduction valve 1
x_2	q_{27}	0	2	steam flow from second steam bleeding of turbine 5
x_3	q_{28}	0	2	steam flow from second steam bleeding of turbine 4
x_4	q_{29}	0	2	steam flow from third steam bleeding of turbine 5
x_5	q_{30}	0	2	steam flow from third steam bleeding of turbine 4
x_6	q_{35}	0	1	steam flow through reduction valve 2
x_7	q_{36}	19.8	24.2	steam flow powering turbine 1
x_8	q_{37}	9.9	12.1	steam flow powering turbine 2
x_9	q_{38}	9.9	12.1	steam flow powering turbine 3
x_{10}	q_{44}	21.15	25.85	steam flow from turbine 4 output
x_{11}	q_{47}	0	1	steam flow through reduction valve 3
x_{12}	q_{51}	16.2	19.8	steam flow from first steam bleeding of turbine 4
x_{13}	q_{52}	0	10	steam flow from first steam bleeding of turbine 5
x_{14}	q_{54}	0	4	steam flow from turbine 1 taken to reduction valve 4
x_{15}	q_{58}	0	4	total steam flow through reduction valve 4
x_{16}	q_{81}	31.5	38.5	water flow to boiler 5
x_{17}	q_{83}	16.2	19.8	water flow to boiler 2
x_{18}	q_{84}	15.3	18.7	water flow to boiler 3
x_{19}	q_{85}	16.2	19.8	water flow to boiler 1
x_{20}	q_{96}	13.77	16.83	steam flow from turbine 5 to collector (element #43)
x_{21}	q_{112}	13.77	16.83	adjustable ‘industrial steam’ flow

Table 2.1: Decision variables for the problem of industrial power plant set-point optimisation.

Decision variable name	Internal model variable name	Lower bound	Upper bound	Description
x_1	q_{22}	0	5.55	flow from first steam bleeding of turbine 1
x_2	q_{23}	0	5.55	flow from first steam bleeding of turbine 2
x_3	q_{24}	0	8.5	flow from second steam bleeding of turbine 2
x_4	q_{25}	0	8.5	flow from second steam bleeding of turbine 1
x_5	q_{26}	50	60	steam flow from turbine 2 output
x_6	q_{27}	50	60	steam flow from turbine 1 output
x_7	q_{28}	0	5	steam flow to heat exchanger
x_8	q_{29}	0	5	steam flow to heat exchanger
x_9	q_{35}	48	80	water flow to boiler 1

Table 2.2: Decision variables for the test problem carved out of the original power set-point optimisation problem (cf. Table 2.1).

some inequality modelling formulae as equalities.) The original model input is a vector of 125 variables. Out of them, 21 inputs were adopted for decision variables vector $\mathbf{x}$; the rest are assigned fixed values.

For the preliminary stage of research a simpler model has been created, which is basically a part of the original model. The original model will be since then referred to as 'plant model' and the simplified one — as 'test model'. Test model properties are to be recognised first, and the plant model is to be approached with the rough methodology and experience gained from the test model; that is why it is right to present here both models.

The test model consists of two coal boilers powering, through a collector, two three-stage turbines. The regeneration block is simplified, but still there are three steam receptions for industrial purposes. The optimisation problem is defined analogously to the one for plant model, with the exception for decision variables whose specification is given in Table 2.2 and the absence of pumping costs. The diagram of the test model is presented in Fig. 2.3.

The model solver was implemented in Fortran language and equipped with file interface. Since it was designed to assist a human operator, the solving algorithm runs interactively, e.g. reporting excessive number of iterations while running (2.2), and asking for a decision. As it was mentioned above, the way the model solving is designed and implemented makes it possible to select such decision values that the output vector $\mathbf{y}$ satisfying (1.3) cannot be found. Such behaviour was not expected and came out only in the course of work for coupling both models with the optimisation module. However it is desirable to reveal certain facts

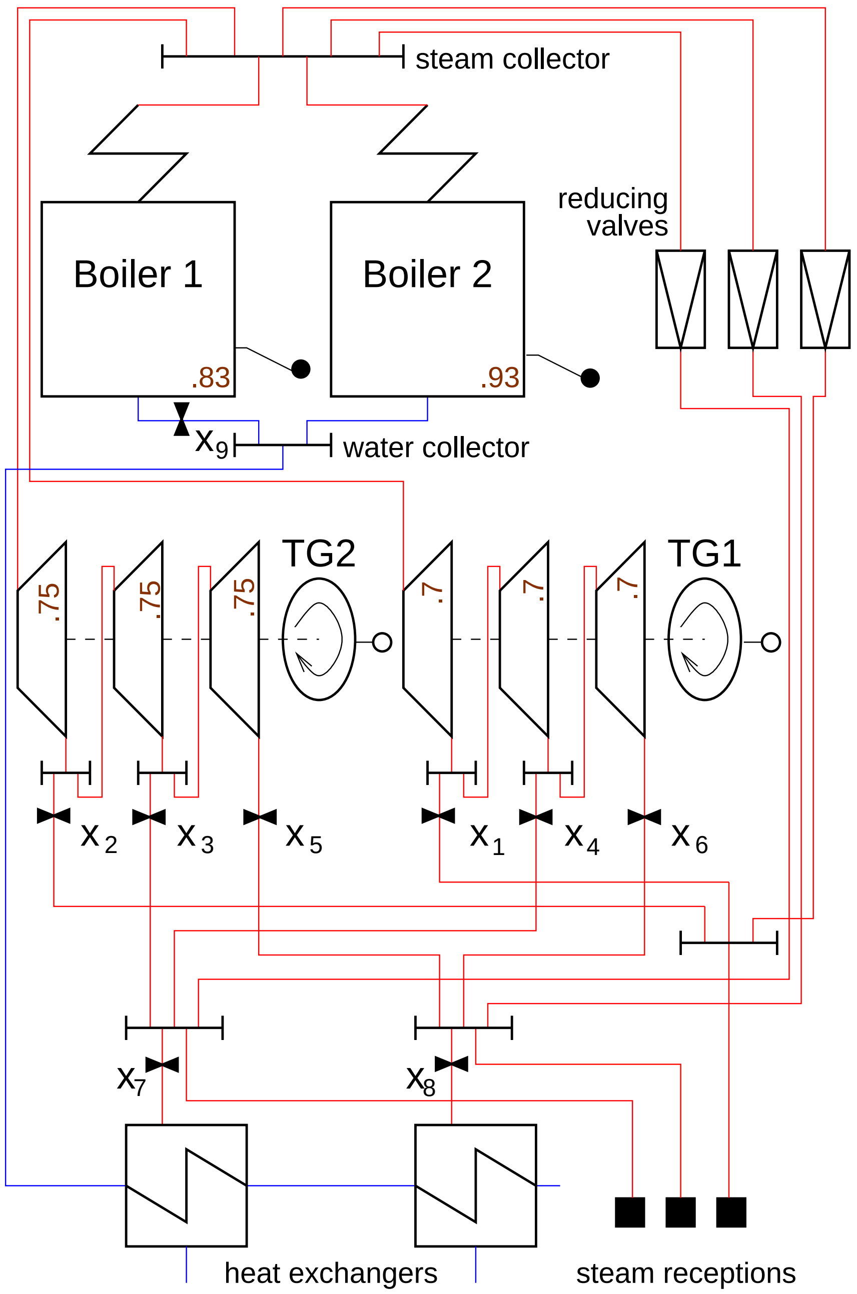


Figure 2.3: The diagram of test power system that is a simplified version of the plant model. The diagram legend is as in Fig. 2.2.

already in this section.

After a deeper model examination it turns out that the reasons for the simulation failure can be twofold. The first are unreachable termination criteria for model internal loops. Normally, equations in (2.1) are constructed so that the actual value $\tilde{\mathbf{r}}(\tilde{\mathbf{q}})$ gives a hint by what value some elements of $\mathbf{q}$ should be adjusted to lessen the imbalance. Apparently, for certain $\mathbf{x}$'s those signals are inadequate to the needed changes of $\mathbf{q}$. The second reason are the mathematical formulae constituting the model — particularly the formulae for steam flow capacity of turbine [75, pp. 72, 125], where there may appear invalid arguments to square root functions, causing the software to report a numerical exception.

A foreseeable potential problem is that for those cases when the simulation is successful, the properties of performance index $f(\cdot)$ based on simulation output are not good enough to attack the problem with wide range of optimisation routines. This is because a model solution is not exact but approximated in a series of iterations. Such approach, with nonzero error margin ϵ is prone to introduce sort of harshness to hypersurface of $f(\cdot)$.

The following paragraphs present the adverse effects that failures and implicit constraints have on the search domain, and the adverse effects iterative solving procedure along with model internal switching have on the performance index itself.

Search domain restricted in two ways

In the course of research for a robust and efficient approach to solve the problem it turned out that implicit and, most of all, feasibility constraints adversely affect the set of feasible decisions. As the result, some optimisation methods designated for this task have to be discarded, and some others have to be adapted. Particularly, attempts to estimate performance index gradient at various points chosen randomly from $D_{\mathbf{x}}$ revealed the importance of feasibility constraints.

For the sake of speed and convenience of lesser dimensionality, gradient estimation test have been performed initially for the test problem as the one containing all the characteristic and troublesome elements, representative for full models: turbine, boiler, heat exchanger. The tests were performed for decisions taken randomly with uniform distribution from $D_{\mathbf{x}}$ as well as in the neighbourhood of some suboptimal initial working-point, selected manually and proposed by IHE. In cases where gradient estimation procedure (based on finite differences of $f(\cdot)$ in the neighbourhood of $\mathbf{x}$) detected anomalies, a number of two-dimensional cuts trough $D_{\mathbf{x}}$ at that point, in versor directions, were generated. Such a typical two-dimensional slice of the decision domain in the proximity of the optimum is presented in Fig. 2.4. Cuts were made by methodical invocation of simulation routine for points across the cutting plane, with purpose to register areas where (1.2, 1.5) or just (1.3) are violated. The areas visible in Fig. 2.4 are

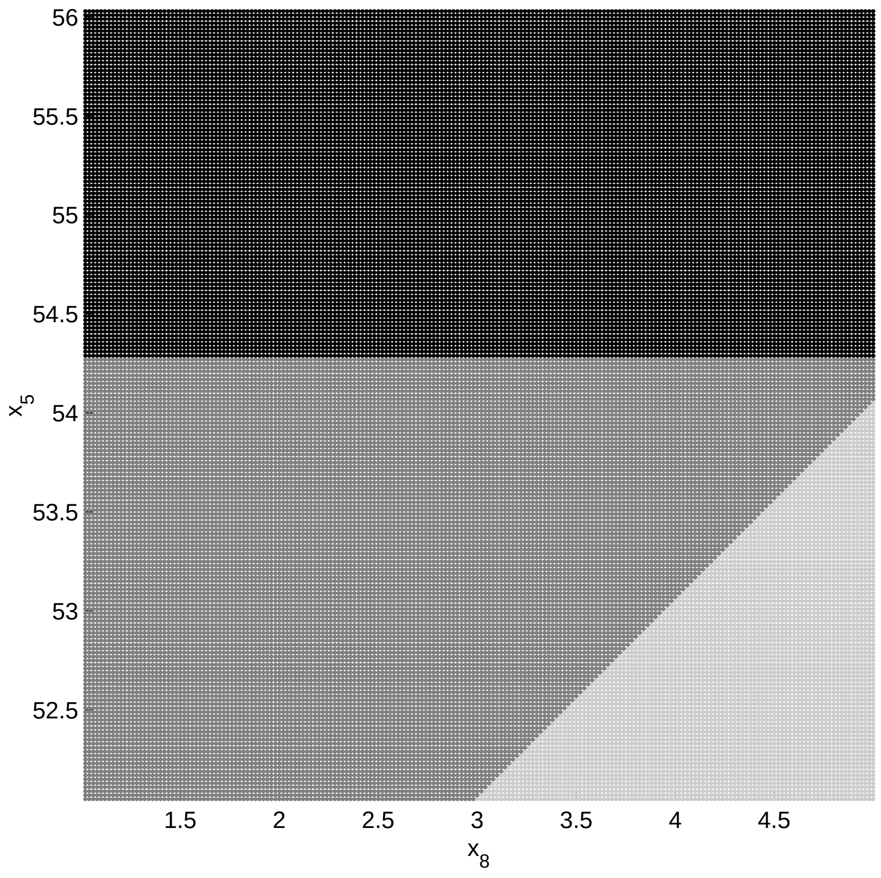


Figure 2.4: A cut through $D_{\mathbf{x}}$ in the neighbourhood of the test problem solution. Area where all constraints are satisfied is marked with light gray colour. Area where implicit constraints are violated is marked with dark gray colour. Area where feasibility constraints are violated is marked with black.

made of overlapping dots, and dot colours represent simulation result. The figure was drawn deliberately as raster and not as contour plot to emphasise the discrete nature of sampling performed over $D_{\mathbf{x}}$. What one can see is that the considerable part of the cut is composed of $\mathbf{x}$'s that violate implicit and feasibility constraints. Fortunately, the gray area where implicit constraints only are violated lies like a buffer zone that coats the region where simulation fails. Moreover, the borders between areas are seemingly straight and potentially supportable by gradient projection or repair routines of optimisation algorithms. As regards both observations, more profound search for anomalies revealed that areas generated by feasibility constraints are not always that straightforward.

The more profound search for anomalies in whole $D_{\mathbf{x}}$ consisted in attempts to estimate gradient at 100,000 trial points chosen randomly from whole $D_{\mathbf{x}}$. The gradient estimation procedure reported feasibility constraints active in 1500 out of those 100,000 cases, which makes the ratio of 1.5%. Considering the number of samples against the model dimension, this ratio is not very reliable, but it confirms the fact that feasibility constraints really get activated and that encountering a region where the simulation fails is quite probable for a single optimisation run. The regions that feasibility constraints may generate in the test problem case can turn out to be unexpectedly complicated, on the contrary to what Fig. 2.4 suggests. Some of them are presented in Fig. 2.5. First of all, they can split $\mathbf{x}$'s lying on an arbitrary plane cutting $D_{\mathbf{x}}$ into several unconnected subsets, as presented in Fig. 2.5a–c. Those subsets are convex and have straight borders, but one cannot assure their shape or unconnectedness in general, in all nine dimensions of the test problem. The black areas of violated feasibility constraints may possibly vanish in some neighbouring parallel plane, or may get broken as shown in Fig. 2.5d. Another conclusion emerging from that particular figure is that the black regions do not have to have straight borders; the broken line in the considered graph is slightly warped downwards. Two more graphs, in Fig. 2.5e–f, reveal other types of singularities introduced to $D_{\mathbf{x}}$ by feasibility constraints. They are just separate points, or curves composed by them, that appear periodically. Their regular shape may be connected in some way with the iterative nature of the simulation process and its stopping criteria, although their apparent zero-measure is inexplicable.

Surprisingly big potential of the test problem to complicate optimisation process is a good excuse for extensive checking of plant model constraints. Similar gradient estimation attempts were made and — naturally enough — simulation failures occurred. Results of the tests are presented in analogous form in Fig. 2.6. They bring two new important observations. The first is that the feasible region may be bounded directly by the feasibility constraints, with no 'buffer space' made of implicit constraints in between (cf. Fig. 2.6a–b). The second observation

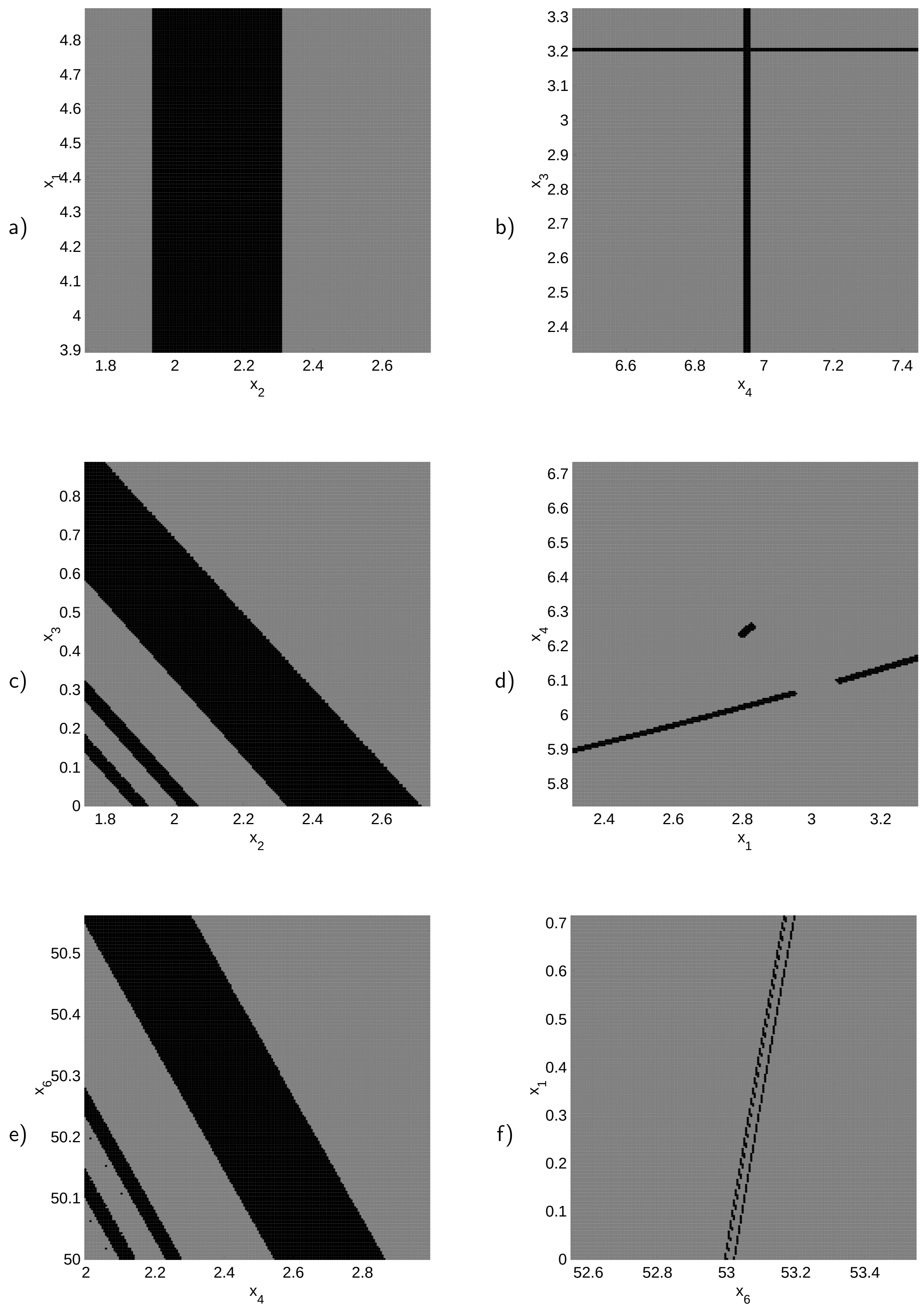


Figure 2.5: Other cuts of search domain $D_{\mathbf{x}}$ where implicit and feasibility constraints are violated. The meaning of colours is the same as in Fig. 2.4.

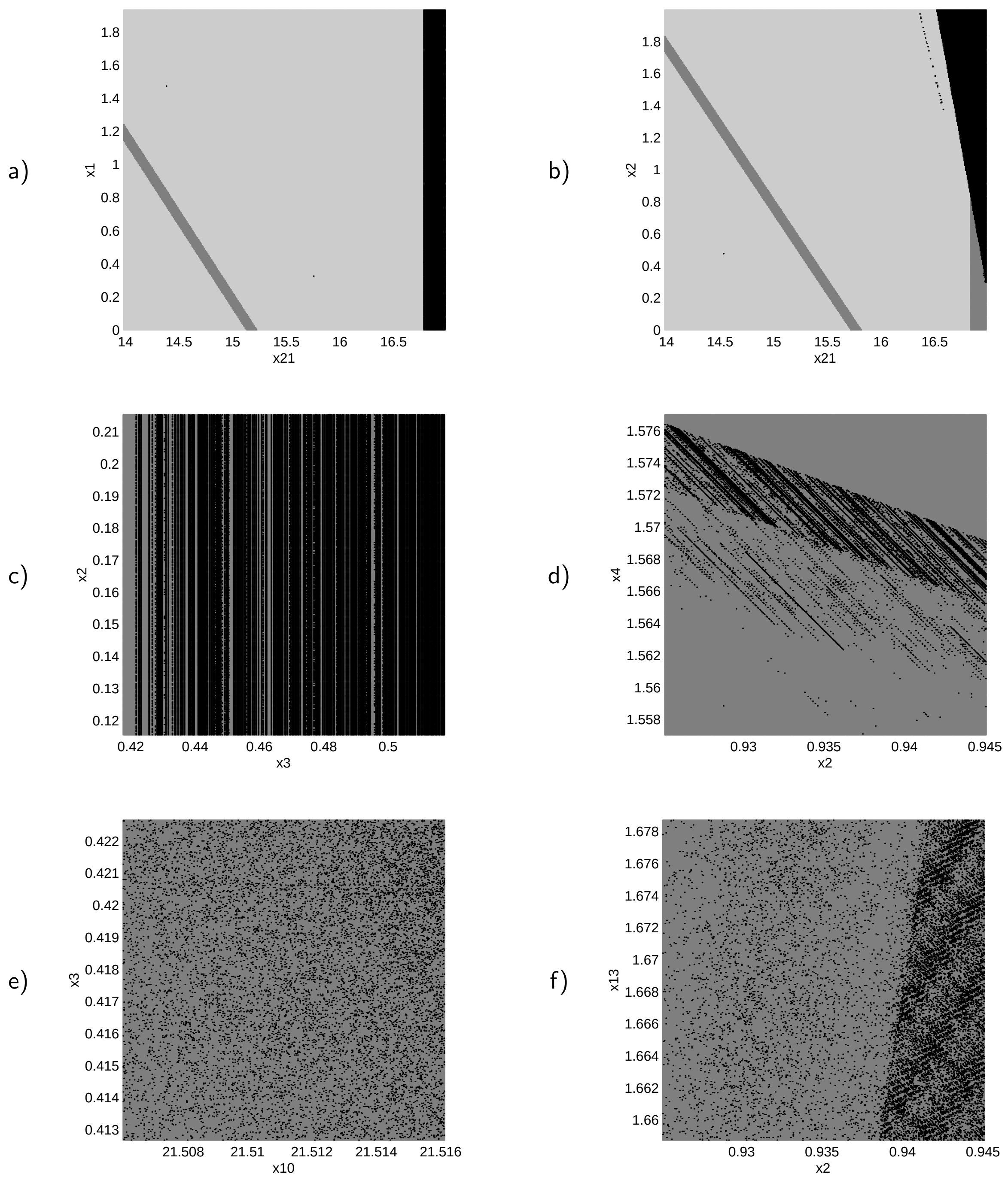


Figure 2.6: Exemplary cuts of search domain $D_{\mathbf{x}}$ with regions of active constraints of various types drawn using colours as in Fig. 2.4. Sections are drawn for some of randomly chosen trial points from $D_{\mathbf{x}}$ where feasible region convexity was detected.

is that the apparent zero-measure isolated areas of simulation failures (cf. Fig. 2.5c–f) are — or can be — 'infinitely many', thus making feasible region like a sieve.[4] Constraints of this type are extremely difficult to handle.

There emerge two consequences of simulation failures for the process of design or adaptation of an optimisation routine. The first one is strictly practical — one has to restore order to the computing environment after such an abortive action initiated by the simulator itself, or by the routine controlling simulations. The second one is the fact that no $\mathbf{y}$ altogether is produced — there is no qualitative measure of by how much the feasibility constraints were violated.

Iterative solving and its impact on performance index surface

Apart from its domain shape, the behaviour of performance index itself is troublesome in certain regions. Those inconveniences have turned up again during gradient estimation tests as well as in operation of non-gradient procedures. The common problem that appeared was relatively big fluctuation of $f(\cdot)$ in small neighbourhood of some $\mathbf{x}$, which made impossible gradient estimation or satisfaction of optimisation routine termination criterion. Fig. 2.7 presents an exemplary plot of performance index $f(\cdot)$ in a particular direction $\mathbf{d}$, i.e. $f(\mathbf{x}_0 + c\mathbf{d}, \mathbf{s}(\mathbf{x}_0 + c\mathbf{d}))$, visualising those fluctuations. The considered range for changes in $f(\cdot)$ and in $\mathbf{x}$ is small, and the picture presents rather a close-up of the surface of $f(\cdot)$. From the engineering point of view, so small fluctuations of $f(\cdot)$ are irrelevant; nevertheless they are there, showing no regularity, unlike in Fig. 2.5f. As it is, they may handicap gradient estimation or optimisation algorithms, until they get discovered as in Fig. 2.7. However, even in case of such such nice graph of fluctuations, there is nobody to decide about some general parameter value for $f(\cdot)$, like Lipschitz constant, in whole $D_{\mathbf{x}}$.

Another exemplary directional performance index graph, presented in Fig. 2.8, shows that those small fluctuations may be accompanied by others, bigger by an order of magnitude. Such big jumps of $f(\cdot)$ are nothing uncommon for the considered model, and may happen also in the area of most interest, i.e. in proximity of the solution (which is the case for Fig. 2.7 and Fig. 2.8). A sudden decrease of the performance function by nearly 0.0014 (in other words, by almost 0.1%) experienced near the value of $7.5 \cdot 10^{-4}$ on the abscissa is a value rarely negligible in case of costs for low-margin industries, and definitely distinguished by most optimisation

[4]One may dare say that feasible sets defined by (1.3) may easily exhibit fractal nature. Consider that one of reasons for simulation failure is infinite loops that happen for certain $\mathbf{x}$'s. In other words, those infinite loops represent badly handled instabilities of simulator internal sets of numerical equations — and such stability criterion is the classic way fractal sets are defined [85, pp. 120–123].

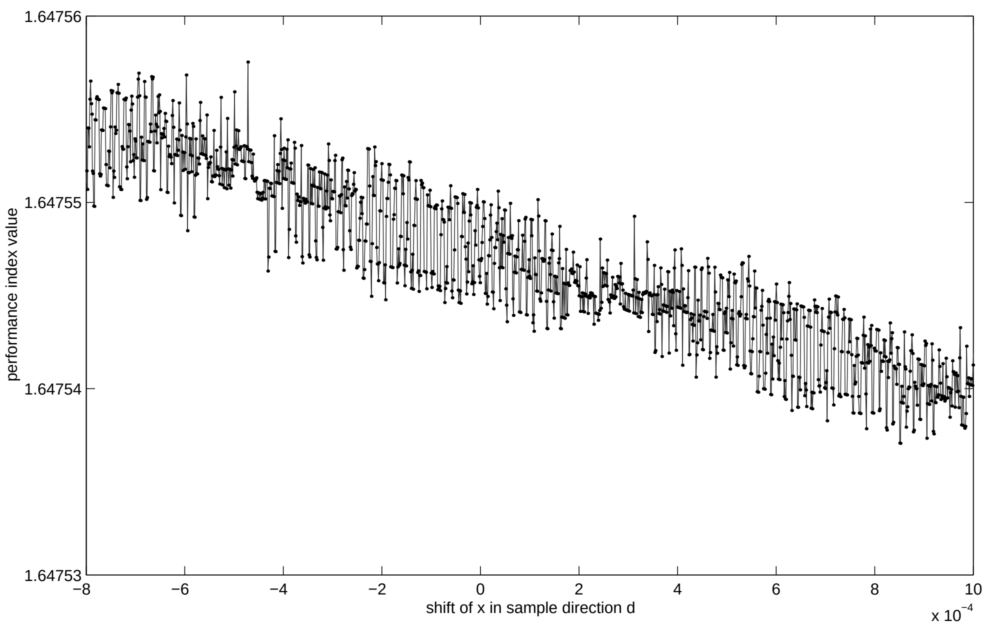


Figure 2.7: Graph of performance index in some direction $\mathbf{d}$, computed close to the problem optimum. Function values for subsequent arguments are marked with dots, the lines are drawn only to indicate dots ordering.

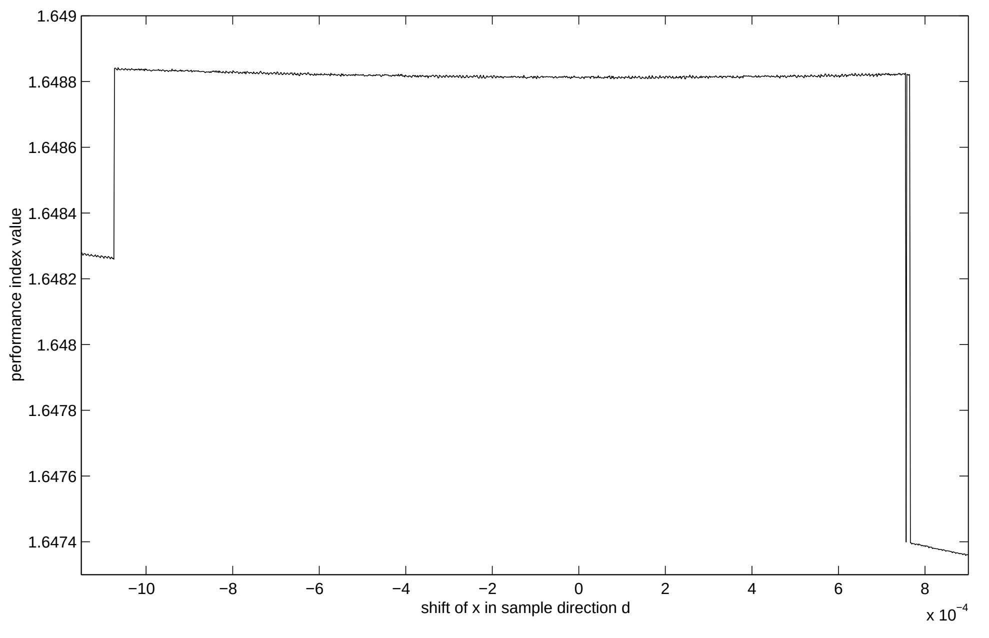


Figure 2.8: Directional graph of performance index, computed close to the optimum. Stepwise character and sudden fluctuations of $f(\cdot)$ are shown.

routines. In the author's opinion, jumps like this one may be effect of some internal switching taking place in the model, for example in case when the steam changes its state. The formulae responsible for computing the model in two adjacent sub-domains may be not 'matched' at the border, not in physical but in numerical sense. This is only a hypothesis waiting for confirmation from some authority in the field. However, in the meantime (as happens in most cases when the product deployment deadlines urge) one has to work with the model as it is, and to adapt the optimisation routine to model specifics in order to ensure stable operation.

2.2 Microwave guide model

This section presents a problem being an essential part of the process of a microwave circuit design, namely the adjusting of waveguide dimensions with purpose to obtain an element of desired electromagnetic properties. Difficulties in solving this type of problems have been signalled from The Institute of Radioelectronics at Warsaw University of Technology (IR).

Before the computer era, calculation of electromagnetic properties of various objects was based on analytical methods, and therefore was applicable to simple devices. Coils, plates, wires can be examples calculable using circuit theory formulae, with reasonable modelling accuracy. For more complex structures those formulae are not sufficient and one has to resort to numerical solutions. Nowadays, given the computer speed, advanced electromagnetic field behaviour simulators can be constructed and harnessed in computer aided design (CAD) of a circuit of high complexity.

The algorithm those simulators use is the Finite Difference Time Domain (FDTD) method. The method was presented first in [113]; it computes the behaviour of electromagnetic fields inside a physical object as the function of time. To make this possible numerically, the physical object as well as the time domain have to be discretised. The object has to be divided logically into cells made of homogenous material. The time is discretised into quants. Next, Maxwell equations can be solved iteratively for each cell and for each time quant.

The simulation algorithm utilises discretised Maxwell equations that determine electromagnetic field distribution and evolution, in the following way. The temporal increments of electric field component in interval $< t, t + \Delta t >$ are calculated from spatial derivatives of the magnetic field at the instant $t + \frac{1}{2}\Delta t$. Analogously, increments of the magnetic field component between $< t + \frac{1}{2}\Delta t, t + \frac{3}{2}\Delta t >$ are calculated from spatial derivatives of the electric field at $t + \Delta t$. This constitutes one step of the simulation procedure.

FDTD method, when applied to properly constructed and adequately excited model, may become a very powerful tool. It is able to calculate field (and heat) distributions, impedances,

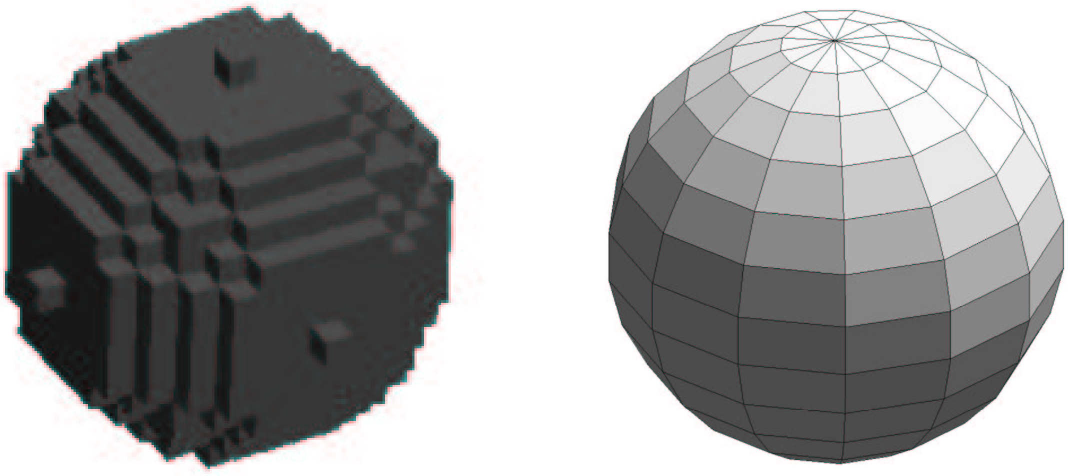


Figure 2.9: Spherical element approximation in original FDTD (left) and in *QW-3D* (right).

frequency-domain characteristics etc. Its applications can be very many as well in civil as in military design tasks, especially under extra preconditions as to designed element size or shape. Thus, to give some examples, one can calculate heating process in a microwave oven, or one can go for calculation of radiation patterns of a cellular phone antenna in vicinity of a human head. Finally, one can consider making a flat antenna, a filter or a resonator of desired properties, on a three-dimensional arbitrary surface, like window pane or a helmet.

Modelling with *QW-3D* package

The FDTD-based simulation software that IR works with is *QW-Simulator*. It constitutes the core of a commercial modelling, simulation, analysis and optimisation package for electromagnetic elements, bearing the brand name *QuickWave-3D* (*QW-3D*). For the detailed information on the product, see [93].

QW-Simulator added value to the original FDTD method is built up by numerous improvements concerning the scope of computable element shapes, simulation efficiency and accuracy. The major limitation of FDTD that has been overcome was the discretisation procedure dividing space domain in cubicoid cells of equal size and made of homogenous material. *QW-3D* allows each cell to be filled with two different materials, the border between them being an arbitrary plane. This reduces time and memory needed for simulation, and simulation errors. The original and improved discretisations are shown in Fig. 2.9, illustrating an exemplary design of a spherical object. Another *QW-3D* facility is to let user define singularities in three-dimensional domain, like infinitely thin wires or surfaces.

QW-Simulator models are stored in files prepared with *QW-Editor*, a program for graphical manual design of elements. While designing, the user may combine a number of pre-defined elements with those created manually and with those imported from a couple of supported CAD programs. Also, *QW-Editor* allows to adjust manually space discretisation patterns in order to indicate regions demanding particular simulation precision. It is also the interface to the simulator through which system excitations and measurement points are defined.

To calculate frequency-domain system characteristics, the system is excited with a pulse on its input. Propagation of this pulse is then calculated in time domain until the effects of the excitation are small enough to stop simulation. The time-domain response can subsequently be mapped into frequency domain by Fourier transform. To compute field distribution at certain frequency, the system is excited with a sine wave. In both cases one has to care about proper excitation signal parameters and sufficiently long simulation run to obtain reliable (i.e. low-noise) results.

Optimisation in *QW-3D*

QW-Simulator is accompanied in *QW-3D* by the optimisation module, *QW-Optimizer* [94], providing an algorithm for automated design of elements. *QW-Optimizer* should be considered rather as a support tool in element design, and not as an autonomous design tool allowing element creation from scratch. It is used only after the user has decided about the element topology and structure, and has provided some element initial dimensions. Powell optimisation routine (described in detail on p. 70) in *QW-Optimizer* uses those initial dimensions as the starting point, and tries to improve the performance index of the element.

Performance index can be defined in *QW-Optimizer* for any parameter calculated in frequency domain (e.g. impedance, scattering matrix). If we consider all parameters calculated by *QW-Simulator* for frequencies of our interest to be the model output $\mathbf{y}$, then the performance index, as it is used by *QW-Optimizer*, can be written as follows

$$f\left(\mathbf{x},\mathbf{y}\right) = \left\|\mathbf{r}\left(\mathbf{a_x},\mathbf{x},\mathbf{x}_\mathrm{L},\mathbf{x}_\mathrm{U}\right)\right\|^2 + \left\|\mathbf{r}\left(\mathbf{a_y},\mathbf{y},\mathbf{y}_\mathrm{L},\mathbf{y}_\mathrm{U}\right)\right\|_{L_p} \quad . \tag{2.5}$$

Therefore, $f(\cdot)$ is made of the sum of two norms of vector functions $\mathbf{r}(\cdot)$ penalising violation of $D_\mathbf{x}$ or $D_\mathbf{y}$. The former norm is, by convention, Euclidean, squared. The latter norm $\|\cdot\|_{L_p}$ for an even $p \geq 2$ is defined for some vector $\mathbf{v}$ as follows

$$\|\mathbf{v}\|_{L_p} = \begin{cases} \max_{i\in\{1,\ldots,\dim\mathbf{v}\}} |v_i| & \text{when} \quad p=\infty \quad , \\ \sqrt[p]{\sum_{i=1}^{\dim\mathbf{v}} v_i^p} & \quad \text{else.} \end{cases} \tag{2.6}$$

The function $\mathbf{r}\left(\mathbf{a_z},\mathbf{z},\mathbf{z}_\mathrm{L},\mathbf{z}_\mathrm{U}\right)$ of some $\mathbf{z}$ and the corresponding vectors of scaling factors $\mathbf{a_z}$, and of lower and upper bounds $\mathbf{z}_\mathrm{L}$, $\mathbf{z}_\mathrm{U}$, returns a vector of weighted constraint violations (if they happen) according to the formula

$$r_i = \begin{cases} a_{\mathbf{z},i}\left(z_{\mathrm{L},i} - z_i\right) & \text{when} \quad z_i < z_{\mathrm{L},i} \ , \\ 0 & \text{when} \quad z_{\mathrm{L},i} \le z_i \le z_{\mathrm{U},i} \ , \\ a_{\mathbf{z},i}\left(z_i - z_{\mathrm{U},i}\right) & \text{when} \quad z_i > z_{\mathrm{U},i} \ , \end{cases} \ . \tag{2.7}$$

Performance definition as in (2.5, 2.6, 2.7) makes it possible to specify different bounds on any parameter of interest in the range of frequencies the simulator is set for. In particular, one may define typical design tasks like making a band-pass filter as optimisation problems. Also, by introducing penalty functions $\mathbf{r}(\cdot)$ for the design and output variables, one makes the optimisation problem an unconstrained one. Furthermore, by avoiding L_p norm of high p, one makes the problem treatable by simple and efficient Powell method.

The above formulation of the performance index that extensively uses exterior penalty functions may provoke objections from the theoretical point of view. In fact, it allows solution of the problem to violate constraints on $\mathbf{x}$ as well as on $\mathbf{y}$. Of the norms available in construction of $f(\cdot)$, the most efficient in penalising tiny violations of $D_\mathbf{y}$ is $\|\cdot\|_{L_\infty}$. It is therefore the most desirable one to be used in problem definition. On the other hand, it introduces sharpness at the points where penalty is activated — the sharpness barely manageable by Powell routine.

In presence of such difficulties *QW-Optimizer* user is presented with the following methodology of attacking the design problem:

1. Prepare the initial design as good as possible with your experience, and theoretical apparatus.
2. Choose correct decision variables, i.e. those practically changeable and really affecting electromagnetic parameters of the element.
3. Start optimisation with small p (4 to 8) — this improves convergence, because the surface of $f(\cdot)$ is smoother, at the cost of solution accuracy.
4. Increase p in later stages of optimisation to achieve higher accuracy.
5. Since $f(\cdot)$ may have local minima, try running the optimisation process from various initial points (designs).

Optimal microwave circuit design problem

The above procedure mitigates inconveniences introduced by implicit constraints and penalty functions enough to make it possible to apply Powell optimisation routine. It does not, however, overcome two other obstacles: local minima and slowness of the optimisation process.

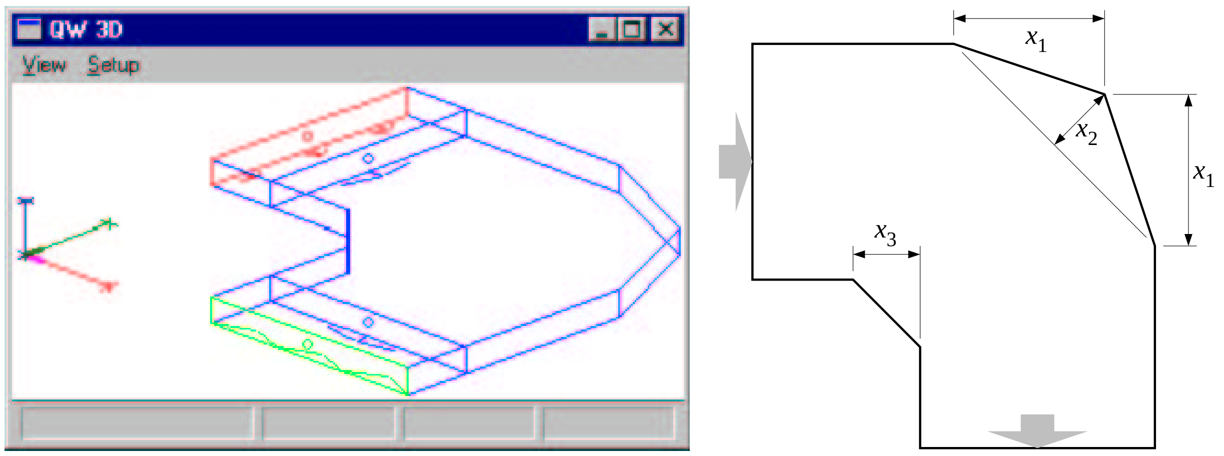


Figure 2.10: Three-dimensional (left) and top (right) view of the waveguide bend subject to design optimisation. The dimensions x_1, x_2 and x_3 are the selected decision variables.

The task undertaken jointly by the author and IR staff was to apply another optimisation routine and to develop alternative, not so restrictive, problem solving methodology. The goals were to:

- Make it possible to carry out optimisation with $\|\cdot\|_{L_\infty}$ norm. This requires implicitly that the optimisation routine should be able to cope with non-smooth surfaces of the performance index $f(\cdot)$.
- Perform search for global optimum rather than the local one. Do it with no need for user involvement in changing the initial optimisation point.
- Make use of the parallel computing environment available in order to speed up the design process. (The Powell routine was incapable of performing computation in parallel in any way.)

The particular design task formulated by IR was setting the dimensions of a waveguide bend. Waveguide is a kind of transmission line, frequently used in gigahertz frequency band. It is a metal pipe of rectangular cross-section, bevelled at its bend as shown in Fig. 2.10. Air is the media the wave propagates through, and propagation direction is indicated with arrows. Bends like this may transmit considerable powers when used e.g. inside a radar. Therefore, one of major requirements is small wave reflection coefficient of such circuit within a specific range of frequencies. It leads directly to formulation of an optimisation problem of waveguide bend design as combination of (1.1) and (2.5). The model output vector $\mathbf{y}$ contains waveguide response in frequency domain. The decision variables $\mathbf{x}$ are specified in Table 2.3. The variable x_3 (*ims*) will be initially not considered as the decision variable. It defines a chamfer used for compensation of so-called fringing field effects which occur on sharp metal edges, and its value

Decision variable name	Internal model variable name	Lower bound	Upper bound	Description
x_1	*ems*	10	30	length of external edge cut
x_2	*mcp*	-5	5	depth of external edge cut
x_3	*ims*	0	12	chamfer width

Table 2.3: Decision variables for the waveguide bend optimal design problem.

can be calculated equally well from theoretical formulae. It will be used to test the impact an increased problem dimension has on optimisation efficiency and efficacy.

As it was indicated above, the original optimisation method used in *QW-Optimizer* does not support well the performance index defined by (2.5) for $p = \infty$. The following two paragraphs discuss in more detail the two major difficulties presented by $f(\cdot)$ — multiple minima and simulation noise.

Implicit constraints mapped into penalty function

The joint work with IR started with examination of the surface generated by $f(\cdot)$ for the case of two-dimensional problem of waveguide bend design. This was done by setting a grid over $D_{\mathbf{x}}$ and calling *QW-Simulator* for each grid point. This effort was done in order to get a closer look on the nature of that surface, and was justified as well by the former IR observations and suspicions concerning the nature of $f(\cdot)$ as by the mere simplicity and small dimensionality of the problem. The surface obtained for whole $D_{\mathbf{x}}$ is presented in Fig. 2.11. There is nothing unusual about its shape, but the values taken by $f(\cdot)$ need a comment. Apparently, $f(\cdot)$ never reaches zero value, which means that no design satisfies the implicit constraints completely. This makes the optimisation problem quite different from that presented in Section 2.1; in that case violating $D_{\mathbf{y}}$ could mean not just imperfect working point but running out of area of model validity. Further explorations of the shape of $f(\cdot)$ in the vicinity of where the optimal point is supposed to lie reveal what makes the optimisation task so difficult.

Fig. 2.12 presents surface view and a contour plot of a function of strikingly steep slopes. Those slopes will actually be discussed in the next paragraph; but there is some other difficulty in $f(\cdot)$ that is visible better on the contour plot — it is two separate attraction areas that have nothing to do with steep slopes. They indicate that there may exist several equally (or nearly equally) good designs for that element. Such behaviour of the performance index explains the need for an optimisation routine of rather global character.

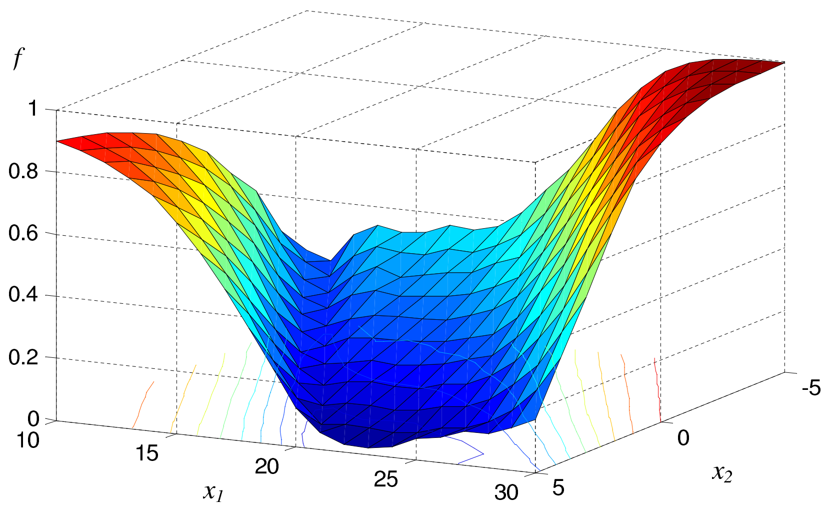


Figure 2.11: View of a surface generated by performance index $f(\cdot)$ over the domain $D_{\mathbf{x}}$ for the two-dimensional problem of optimal waveguide bend design.

Effects of electromagnetic simulation error

Apart from multiple minima, the performance index exhibits features that are the effect of the underlying numerical simulation procedure. They were presented first in Section 2.1; they are occasional large jumps plus ubiquitous small fluctuations of $f(\cdot)$. Both are clearly visible on the surface view in Fig. 2.12. The cause of the fluctuations, as explained in [93, p. 13], is the termination criterion in wave propagation simulation procedure, which makes the simulator stop when the field amplitudes are reduced to negligible values. Due to finite numeric accuracy and accumulated errors, this may happen slightly earlier or later, producing a sort of noise.[5]

The source of sudden jumps in not explained well. Probably it is related to the change in the number of filled cells involved in simulation as element dimensions are changed. It comes from IR staff experience that with finer space discretisation those jumps become more frequent, and simultaneously lower. Such situation is illustrated in Fig. 2.13, where the cross-section through the actual performance index is given with thin black line, and some possible shape of the same performance index for finer discretisation is drawn with thick gray line.

Refining discretisation also affects the simulation error — small fluctuations are significantly reduced. It all comes at the cost of the simulation time and memory usage: reducing the cell size by 2 decreases the error by 4, increases memory usage by 8 and the simulation time by 16. The cell size can be changed only manually, i.e. the optimisation routine has no

[5]Here, the term 'noise', although used customarily in the branch, refers rather to appearance of error rather than to its randomness. Such 'noise' values are completely deterministic — but not determined by any open-form mathematical formula.

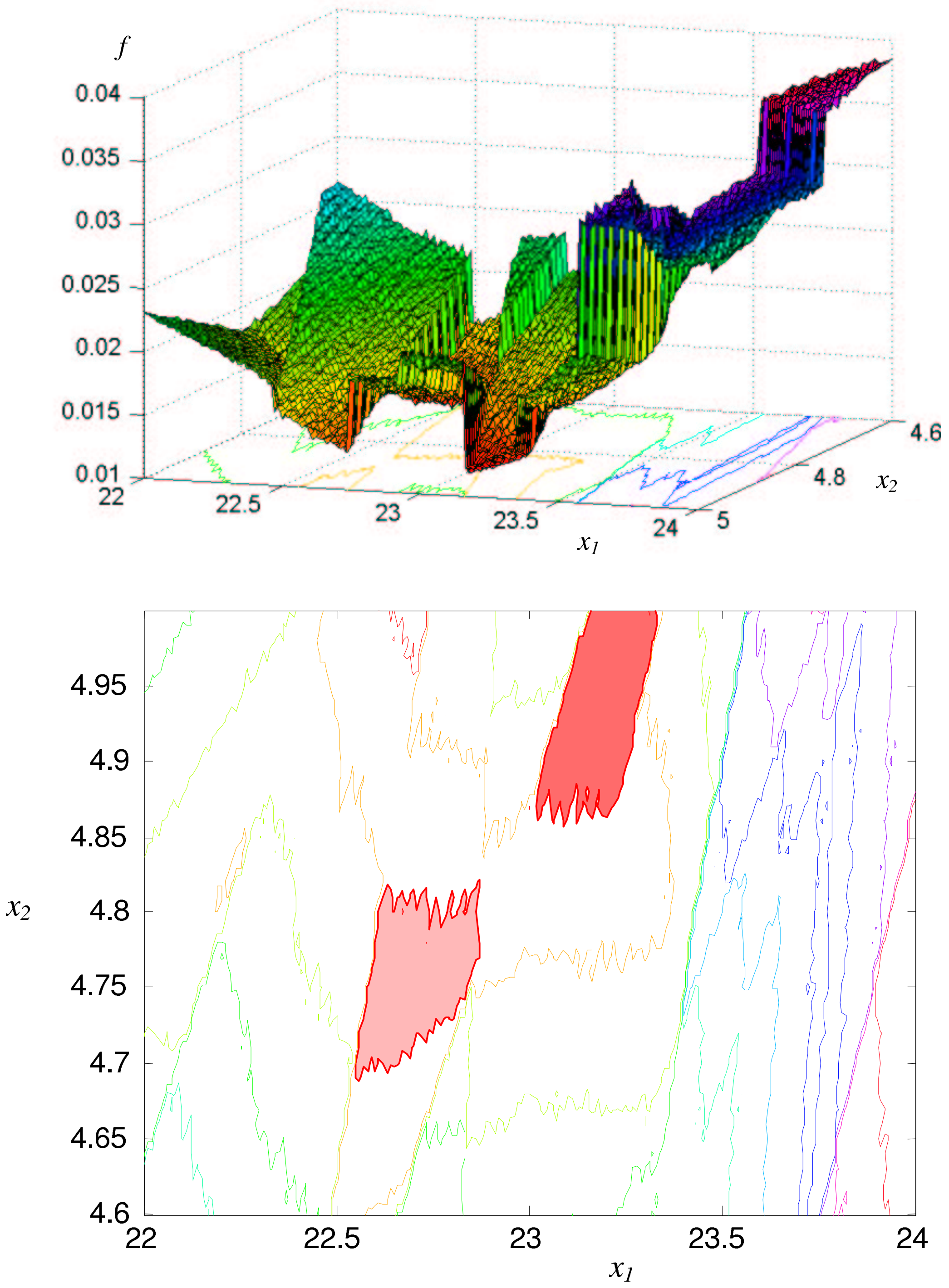


Figure 2.12: View of performance function surface (top) and the corresponding contour plot (bottom) for waveguide bend design optimisation problem. The two filled areas in the bottom graph denote two attraction regions of better (darker colour) and worse (lighter colour) alternative for waveguide designs.

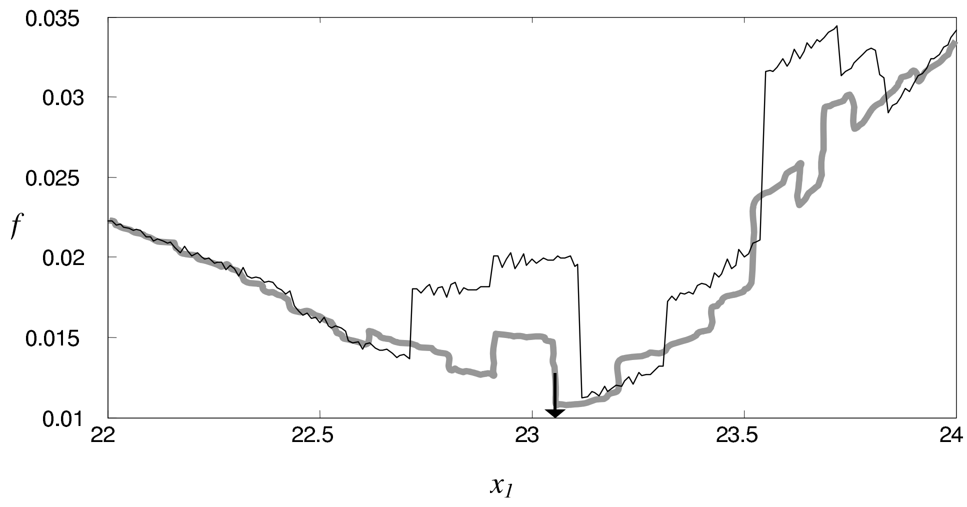


Figure 2.13: Cross-section through performance index (cf. Fig. 2.12) in the optimum neighbourhood for $x_2 = 4.96$. The narrow black line presents the real graph for the actual discretisation. The wide gray line presents what that graph could look like for some finer discretisation.

way to adjust it.

Let us return to Fig. 2.13 to examine the effects of finer discretisation. It turns out that it can affect $f(\cdot)$ so that the minimum (indicated by an arrow) is located where the step plateau was in the case of coarse discretisation. Such dramatic dependence of the shape of $f(\cdot)$ on some simulation parameters breeds distrust for the results produced altogether, and makes one search for some optimisation methods that rely on a set of points only, rather than on every single one. Techniques that could be of interest in such case are presented in Section A.2.

2.3 IP services market model

This section presents the problem of optimal pricing for networking products.[6] Price is — sometimes more than brand, quality, technical support and other factors — the attribute of a product much distinguished in the market game between the seller and the buyer. Probably it is so because of the immediate effect prices have on the most measurable and instant effect of a transaction — the profit. It is also because the price is the attribute easily comparable

[6]In general, pricing can be applied at operational and at marketing levels. At the operational level its main purpose is to prevent network congestion and to enforce some policy of resource distribution — prices of resources are treated more like control signals and not as real-life prices to be paid in true currency. At the marketing level prices are what they are for ordinary people — i.e. money that has to be spent for some product or service. Here we assume the marketing perspective; for more on the operational perspective, see e.g. [67] and references therein; for a concise comparison of those perspectives, see [10].

among competitive products.

The problem of optimal pricing for IP network services was one of the major topics addressed by 'Quality of Service and Pricing Differentiation for IP Services' (QOSIPS) project, run within Fifth Framework Programme of the European Commission. The Institute of Control and Computation Engineering (ICCE), the author's mother institute, was one of five QOSIPS participants. Of QOSIPS products, the Pricing Module (PM) is focused on optimal pricing of IP network services, like Virtual Private Network (VPN), with support for calculation of service utilisation and Quality of Service (QoS) modelling. Being one of PM team members, the author was directly involved in the process of model making and of price optimisation. The market model considered and analysed in this section is not precisely a real-life example. Nevertheless, all the modelling and optimisation was made using PM, and the model parameters are adapted in order to emphasise potential problems one can encounter during optimisation.

In the past few decades much effort was made by economists to construct models of human attitude towards changing price, and the willingness to commit a transaction (see e.g. [103] for the details). Most of those models intend to describe the process of purchasing standard goods by mass customers. The economy models, in order to remain useful, have to be easy to tune and — since some tuning data may not come just from statistics but through customer polls and scenarios made by people of various knowledge of economy — they also have to be understandable by laymen. Let us present four most used ones in the order of growing complexity:

- Linear model. Its advantages are great simplicity and possibility of tuning using standard identification procedures (i.e. by least squares routine). The disadvantages are small validity range ($5 \div 10\%$) and symmetry that discards more violent customer reaction to an increase of price $\mathbf{x}$ than to a decrease. The formula for linear model is

  $$r_{\mathrm{LIN}}(\mathbf{a}, b, \mathbf{x}) = b - \mathbf{a}^T\mathbf{x} \quad , \tag{2.8}$$

  with $\mathbf{a}$ and $\mathbf{b}$ representing coefficients.

- Multiplicative, or Cobb-Douglas, model. It derives from the term of *elasticity*, describing dependence between relative changes of output against relative changes of model input. Cobb-Douglas model is helpful in modelling sales of own products (in such case elasticity is negative) as well as in modelling the impact of competitive product prices (elasticity is positive). Models tune well using the same least squares routine. Cobb-Douglas model shares with the linear model the same inconvenience of limited validity area. The

Cobb-Douglas formula is

$$r_{\mathrm{CD}}(\alpha, \boldsymbol{\beta}, \mathbf{x}) = \alpha \prod_{i=1}^{\dim \mathbf{x}} x_i^{\beta_i} \quad , \tag{2.9}$$

where α is a coefficient and $\boldsymbol{\beta}$ — vector of elasticities.

- Attraction model. It bases on the assumption that the sales are proportional to the attraction of a product as compared to cumulative attraction of all competitive products on the market. There is freedom for formulae to be used for calculating attractions. Attraction model can be applied in wider range of price changes than the two preceding models. However, there is no linear tuning procedure available for it. The attraction model formula is

  $$r_{\mathrm{ATTR}}(\alpha, \mathbf{x}) = \alpha \frac{x_1}{\sum_{i=1}^{\dim \mathbf{x}} x_i} \quad , \tag{2.10}$$

  where α is a scaling factor. It was assumed in (2.10) that the attraction for product i is plainly its price x_i.

- Gutenberg model. It puts emphasis on the phenomenon that customers are insensitive to small changes of prices. Its output is driven by the difference of the changed price and the mean of all relevant prices before the change was made. This allows modelling in broader range than for the first two model types. Gutenberg model drawbacks concern mainly the tuning phase in which nonlinear identification procedures must be employed. The Gutenberg formula is

  $$r_{\mathrm{GB}}(a, b, c_1, c_2, x, \bar{x}) = a - bx - c_1 \sinh\left(c_2(x - \bar{x})\right) \quad , \tag{2.11}$$

  where a, b, c_1 and c_2 are coefficients, x is the changed price of the considered product and $\bar{x}$ is the average price of own and competitive products before the change of x was made.

Effectiveness of the above model types is shown in Fig. 2.14 where each of them is tuned to match a sample market response curve. Of course, one can consider using combination of those models, provided there exist enough data for sufficient model excitation in the tuning phase.

Market modelling in PM

The work on PM market model development was done in QOSIPS jointly with a partner company that was considerably experienced by development of decision support tools. A solid branch of its products are tools for market modelling and optimal pricing. Its success was

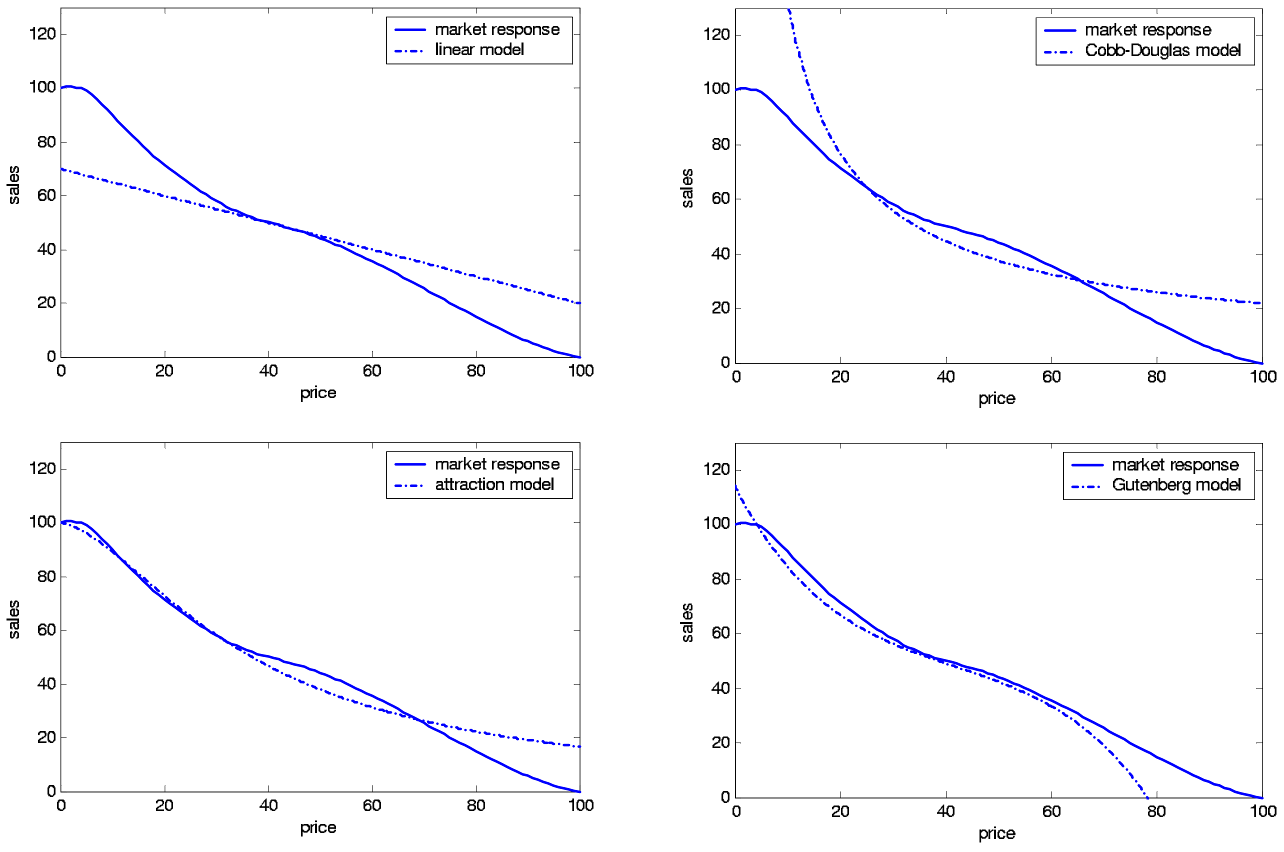


Figure 2.14: Fitting models to a given market response curve. Subsequent graphs illustrate modelling scope and accuracy for linear, Cobb-Douglas, multiplicative and Gutenberg model types. The market response curve is drawn with a solid line, model response curves are dash-dotted.

particularly remarkable when price optimisation was applied for uniform products being sold in big volume, like petroleum, supermarket goods or cellular phones [104].

Knowledge support systems by that partner had much impact on IP services market modelling in QOSIPS. However, the way network service products are constructed and being sold required development of a new model, able to take into account the following facts:

- Products are complex and made as mixture of obligatory and facultative items;
- Products of one kind are sold in relatively small number, and are diversified by the choice of component items (made by the customer) and by the prices (as the result of frequent price negotiations or discounts);
- Unlike for petroleum, the customers are subscribers, which implies obligations on both parties (customers — loyalty, network operator — reliability);
- Customers use the product in two ways: subscribing (once) and utilising (daily);
- By the nature of network, customers share resources and affect their QoS mutually.

The model module, developed and implemented jointly by QOSIPS partners, allows to perform simulation of market behaviour, with all important network performance and economic indicators, within a specific time horizon. The market and network are treated as one discrete-time dynamic system. The state variables are numbers of subscribers for each product i being offered to the market.[7] The decision variables are prices of Network Solution Provider (NSP) own products. They remain constant throughout the simulation. There is neither noise nor uncertainty[8] present explicitly in the system. Let us now present the state and output equations. Here, they have been simplified enough to make the case clearer, without losing any of the problem characteristics important for this dissertation. For the full description of PM market model, refer to QOSIPS consortium deliverables [28, 29].

The number of subscribers at the beginning of time period (usually, month) $t+1$ is denoted by vector $\mathbf{q}_\mathrm{S}(t+1)$ and determined by the state equation

$$\mathbf{q}_\mathrm{S}(t+1) = \mathbf{q}_\mathrm{S}(t) + \mathbf{q}_\mathrm{S}^\mathrm{in}(t) - \mathbf{q}_\mathrm{S}^\mathrm{out}(t) \quad , \tag{2.12}$$

[7]In reality, the model is more complex: it is rooted in correct dividing market into segments of customers that could be characterised by similar attitude towards price changes, and by similar network usage pattern. One product sold in different segments is characterised by different sales model, and by separate number of subscribers. There are more features of PM not covered here; they have been presented in [9].

[8]PM authors from the very beginning were aware of indeterministic nature of almost all phenomena they aimed to model. However, for the sake of simplicity randomness is suppressed at certain points by calculation of expected values. The broader discussion on indeterminism in PM can be found in [60].

where $\mathbf{q}_{\mathrm{S}}^{\mathrm{in}}(t)$ and $\mathbf{q}_{\mathrm{S}}^{\mathrm{out}}(t)$ are volumes of customers who subscribe or renounce NSP services during period t. Volume of those newcomers and leavers is, naturally, the function of price vector $\mathbf{x}$, but also the function of the overall QoS experienced by the users in period t, according to formulae

$$q_{\mathrm{S},i}^{\mathrm{in}}(t) = r_{\mathrm{CD}}(\alpha_i^{\mathrm{in}}, \boldsymbol{\beta}_i^{\mathrm{in}}, \mathbf{x}) q_{\mathrm{Q},i}(t) \quad , \tag{2.13}$$

$$q_{\mathrm{S},i}^{\mathrm{out}}(t) = r_{\mathrm{CD}}(\alpha_i^{\mathrm{out}}, \boldsymbol{\beta}_i^{\mathrm{out}}, \mathbf{x})(1 - q_{\mathrm{Q},i}(t)) \quad , \tag{2.14}$$

where $q_{\mathrm{Q},i}(t)$ is the indicator of QoS experienced by subscribers of product i, and defined as fraction of total data set that were conforming to Service Level Agreement (SLA) made between subscriber and NSP.[9] As is readily seen, standard Cobb-Douglas model (2.9) for sales has been implemented in PM with a term taking into account also QoS deterioration.

Intricacy of network traffic is so big that, in order to model QoS accurately, one should either make strict simplifying assumptions about network topology and traffic, or to resort to network simulators [80, 38]. Both methods aspire to infer about exact network state, either with help of analytical formulae, or by simulations. Both approaches are unacceptable in case of PM because of limited scope of application or numerical complexity. In PM a simple network model has been assumed that aggregates utilisations of selected network services, following the formula

$$q_{\mathrm{Q},i}(t) = \begin{cases} 1 & \text{if} \quad z_i > 1 \\ 0 & \text{if} \quad z_i < 0 \\ z_i & \text{otherwise} \end{cases} \quad , \tag{2.15}$$

$$z_i = a_i^{\mathrm{quality}} + \textstyle\sum_{j=1}^{\dim \mathbf{q}_{\mathrm{S}}} b_{i,j}^{\mathrm{quality}} q_{\mathrm{S},j}(t) q_{\mathrm{F},j}(t) \quad ,$$

where $\mathbf{a}^{\mathrm{quality}}$ and $\mathbf{b}^{\mathrm{quality}}$ contain coefficients and $\mathbf{q}_{\mathrm{F}}(t)$ is the vector of utilisations for network services. This simple scheme makes it possible to indicate (through $\mathbf{a}^{\mathrm{quality}}$ and $\mathbf{b}^{\mathrm{quality}}$) groups of customers that interfere using up the same resources. The saturation clause in (2.15) protects QoS indicator from running out of its scope.

Utilisation $q_{\mathrm{F},i}$ of product i is calculated by classic Cobb-Douglas formula

$$q_{\mathrm{F},i}(t) = r_{\mathrm{CD}}(\alpha_i^{\mathrm{usage}}, \boldsymbol{\beta}_i^{\mathrm{usage}}, \mathbf{x}) \quad . \tag{2.16}$$

(Actually, $\mathbf{q}_{\mathrm{F}}$ remains constant w.r.t. $\mathbf{x}$ and the period index t is left only for completeness.)

Equations (2.12), (2.13), (2.14), (2.15) and (2.16) define state function of the dynamic system, the market model. The numerical procedure executed at every simulation step is

[9] More information on making SLA's and providing QoS in IP networks using standard mechanisms can be found in [115].

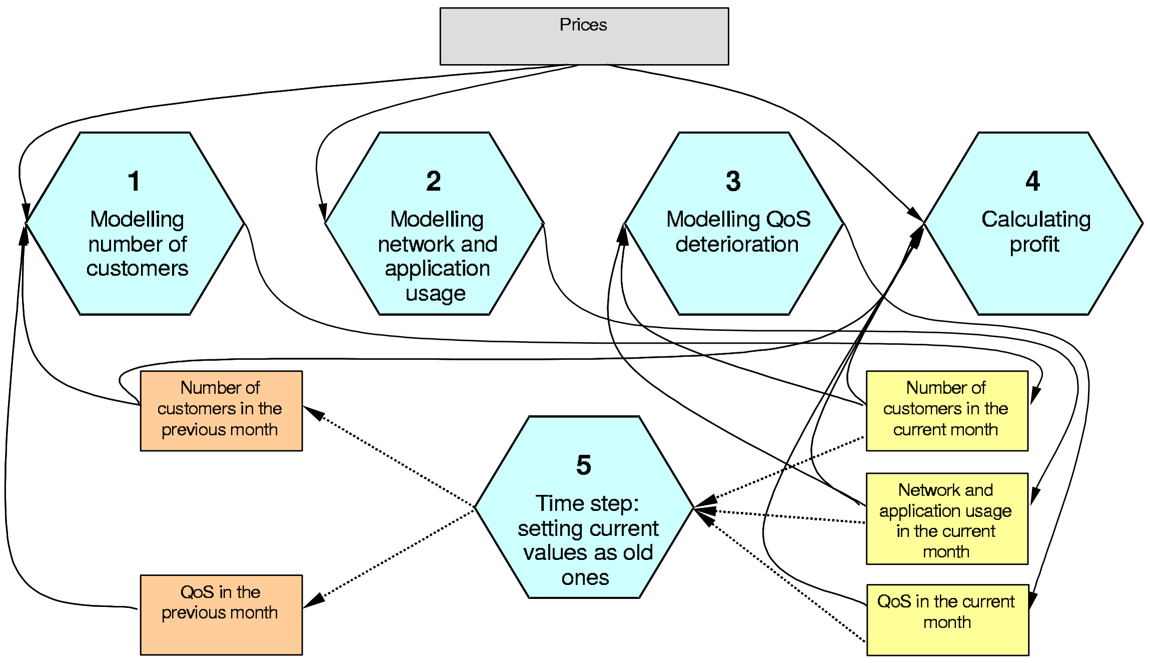

Prices
1 Modelling number of customers
2 Modelling network and application usage
3 Modelling QoS deterioration
4 Calculating profit
5 Time step: setting current values as old ones
Number of customers in the previous month
QoS in the previous month
Number of customers in the current month
Network and application usage in the current month
QoS in the current month

Figure 2.15: Operations (1–5) performed during one simulation step in PM. Arrows indicate data used for each operation.

shown schematically in Fig. 2.15. Of the operations presented there, Operation 4, deserves a comment: it is the calculation of economic indicators, income and costs, that are part of the simulation output vector $\mathbf{y}$. NSP income and incurred costs in time period t are split across all products, and stored in vectors $\mathbf{q}_\mathrm{I}(t)$ and $\mathbf{q}_\mathrm{C}(t)$, respectively. Income and cost are functions of prices and of internal model variables

$$\begin{aligned}\mathbf{q}_\mathrm{I}(t) &= \mathbf{r}_\mathrm{I}\left(\mathbf{x}, \mathbf{q}_\mathrm{S}(t), \mathbf{q}_\mathrm{S}^\mathrm{in}(t), \mathbf{q}_\mathrm{S}^\mathrm{out}(t), \mathbf{q}_\mathrm{Q}(t), \mathbf{q}_\mathrm{F}(t)\right) \quad , \\ \mathbf{q}_\mathrm{C}(t) &= \mathbf{r}_\mathrm{C}\left(\mathbf{x}, \mathbf{q}_\mathrm{S}(t), \mathbf{q}_\mathrm{S}^\mathrm{in}(t), \mathbf{q}_\mathrm{S}^\mathrm{out}(t), \mathbf{q}_\mathrm{Q}(t), \mathbf{q}_\mathrm{F}(t)\right) \quad .\end{aligned} \tag{2.17}$$

The modelling functions $\mathbf{r}_\mathrm{I}(\cdot)$ and $\mathbf{r}_\mathrm{C}(\cdot)$ may have multiple optima w.r.t. $\mathbf{x}$, and may be discontinuous. The simulation output $\mathbf{y}$ are the values of all internal variables in all time periods simulation was run for, plus some their aggregates (e.g. total number of customers and profit calculated as $\sum_{i=1}^{\dim \mathbf{q}_\mathrm{I}} q_{\mathrm{I},i} - \sum_{i=1}^{\dim \mathbf{q}_\mathrm{C}} q_{\mathrm{C},i}$) denoted here by $\mathbf{q}_\mathrm{A}$. For simulation time horizon N the output vector $\mathbf{y}$ can be written as

$$\begin{aligned}\mathbf{y} = (\quad & \mathbf{q}_\mathrm{S}(1), \ldots, \mathbf{q}_\mathrm{S}(N), \mathbf{q}_\mathrm{S}^\mathrm{in}(1), \ldots, \mathbf{q}_\mathrm{S}^\mathrm{in}(N), \mathbf{q}_\mathrm{S}^\mathrm{out}(1), \ldots, \mathbf{q}_\mathrm{S}^\mathrm{out}(N), \\ & \mathbf{q}_\mathrm{Q}(1), \ldots, \mathbf{q}_\mathrm{Q}(N), \mathbf{q}_\mathrm{F}(1), \ldots, \mathbf{q}_\mathrm{F}(N), \\ & \mathbf{q}_\mathrm{I}(1), \ldots, \mathbf{q}_\mathrm{I}(N), \mathbf{q}_\mathrm{C}(1), \ldots, \mathbf{q}_\mathrm{C}(N), \\ & \mathbf{q}_\mathrm{A} \quad) \quad .\end{aligned} \tag{2.18}$$

Optimal product pricing problem

The aspect of QOSIPS that is of our interest here, is utilisation of a properly identified model of IP services market with the objective to maximise NSP profits in N consecutive monthly time periods. Usually, prices in the considered branch are changed every half year, and definitely not sooner than after a quarter. In the model presented here $N = 5$ is assumed, which is a reasonable simulation horizon. As is was mentioned above, the model presented here is much simpler than the original one; also its parameters are modified with purpose to demonstrate how difficult problems may still be created by so simple modelling formulae.

For convenience reasons, assume that $q_{\mathrm{A},1}, \ldots, q_{\mathrm{A},N}$ denote the total number of NSP subscribers in months $1, \ldots, N$, and that $q_{\mathrm{A},N+1}$ is the total profit over all products and all months considered. Then, the optimal product pricing problem is the minimisation of function $f(\cdot)$ defined as follows

$$f(\mathbf{x}, \mathbf{y}) = -q_{\mathrm{A},N+1} \tag{2.19}$$

with the implicit constraints

$$q_{\mathrm{A},t} > q_{\mathrm{S}}^{\min} \ , \quad t = 1, \ldots, N. \tag{2.20}$$

Equation (2.19) requires no particular comments. Although managers often express desire to make market share or the brand perception, or awareness, the extra objectives, such demands are not accompanied by any idea how to measure them, or how to scalarise such multiobjective problem. Equation (2.20) specifies that the total number of NSP subscribers in each month considered cannot fall below a certain level $q_{\mathrm{S}}^{\min}$. Such requirement guarantees NSP some minimum income derived from monthly fees and from other flat charges imposed on subscribers, thus ensuring minimum profitability of the business. Such formulation of the implicit constrains could be replaced by direct requirement for the minimum overall profit. However, this is not done, possibly partially due to management staff habits, and partially because (2.20) secures some minimum initial number of subscribers at the beginning of the next decision period when the prices change.

The considered model consists of two products marked by two prices, x_1 and x_2, that are the decision variables. They are subject to standard constraints (1.4a).[10] Parameters of the model being under consideration are exactly the same for both products. It is assumed that there are no competitive products on the market, and therefore that the number of new and renouncing subscribers depends only on the own price, x_1 or x_2. The sensitivities in formulae for $\mathbf{q}_{\mathrm{S}}^{\mathrm{in}}$ (2.13) and $\mathbf{q}_{\mathrm{F}}$ (2.16) are both common-sense (-1 and -0.2, respectively). The

[10] In practice, the lower bound is of highest interest — for NSP that are non-dominant on the market it is set to the corresponding price of the dominant NSP, reduced by some margin.

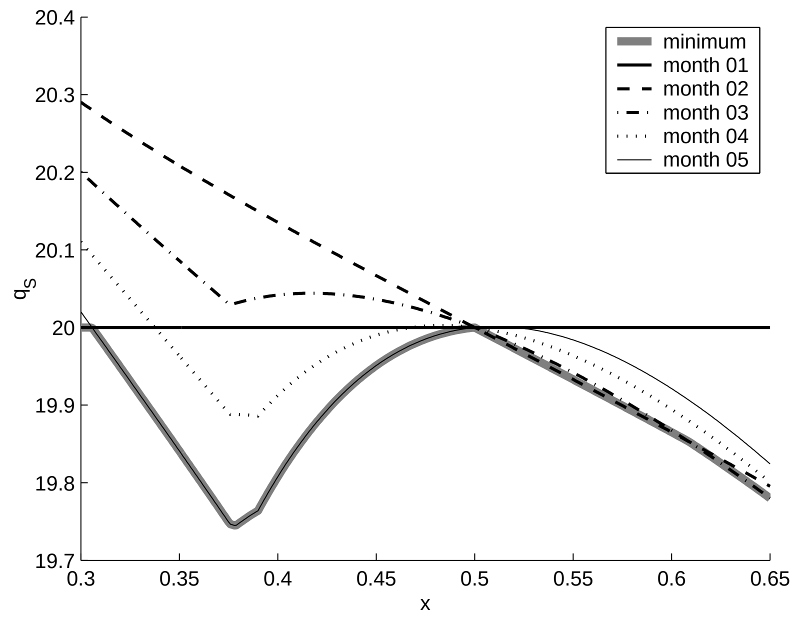


Figure 2.16: Graph of the number of subscribers in the first 5 months as the function of changed price in unidimensional exemplary market model. The initial number of subscribers was 20 and the old price value was 0.5. The minimum observed number of subscribers in all 5 months is drawn with thick gray line.

sensitivity in formula for $\mathbf{q}_S^{\text{out}}$ (2.14) is set to -1. Parameters for (2.15) were selected so that both 0 and 1 values can be reached easily in $D_{\mathbf{x}}$. (This means that NSP network is rather underprovisioned.) The next paragraph presents types of problems such apparently simple model, as well as models alike, is able to create.

Simple models — difficult problems: unconnected domains and multiple minima

The fact that the discussed model is a product of simplification of original PM model does not prevent it from generating non-connected areas of feasible $\mathbf{x}$'s, like those in Section 2.1. This happens again thanks to implicit constraints (2.20). The situation persists also for the model restricted to just one product offered. Fig. 2.16 presents graphs of the number of subscribers vs. price, for subsequent months, with the minimum value drawn by wide gray line. For $x > 0.5$ implicit constraints are violated in the first months by sudden ebb of customers scared off by high price (later, this is balanced by newcomers attracted by better QoS conditions when q_S is low). For $0.37 < x < 0.5$ it is the opposite: setting the price lower first attracts subscribers with that bargain price — only to repel them by worsening QoS, being the effect of network congestion. For $x < 0.37$ QoS indicator reaches zero in one or more months,

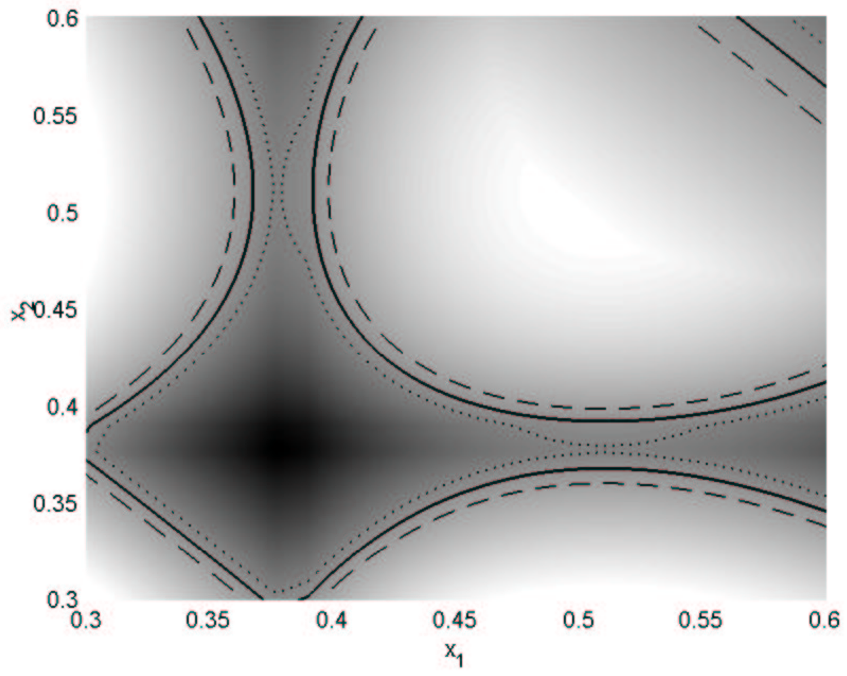


Figure 2.17: Plot of the minimum number of subscribers in the first 5 months for the two-dimensional model. Lighter areas denote bigger number of subscribers. The solid, dashed and dotted lines enclose feasible regions for diverse minimum number of subscribers $q_{\mathrm{S}}^{\min}$.

cutting off newcomers except for those attracted in the initial periods. Now, by setting an imaginary limit of subscribers at 19.8 or 19.9, one can see the domain may be partitioned into two subdomains. Situations like this one happen also when a model of identical structure, but with different parameters, is investigated in steady state (i.e. on the infinite time horizon). Such case is described in more detail in [60].

The same mechanisms are in force for the model with two products, except for the fact that the subdomains do not have to be convex sets. Fig. 2.17 illustrates such case. The areas of $D_{\mathbf{x}}$ where (2.20) is satisfied can be in general not connected (those bounded by solid and dashed lines) but, in particular, also non-convex (the dotted line). One may object that this can be eliminated by setting $D_{\mathbf{x}}$ so that 'unreasonable' regions, i.e. those where SLA is violated, are excluded. This argument is right, except for two cases. The first is that a manager in NSP may initially not *know* what are the reasonable bounds for $\mathbf{x}$, and the decision support tools, like PM, are to make it clear. Such cases should be eliminated by the manager who is to judge the solution produced by a decision support tool. The second is that some other manager may deliberately *want* to exploit the customers by involving them in disadvantageous contracts. Such cases should be eliminated by a properly constructed SLA.

Another issue is the shape of the performance index $f(\cdot)$, the profit. Since PM simulation

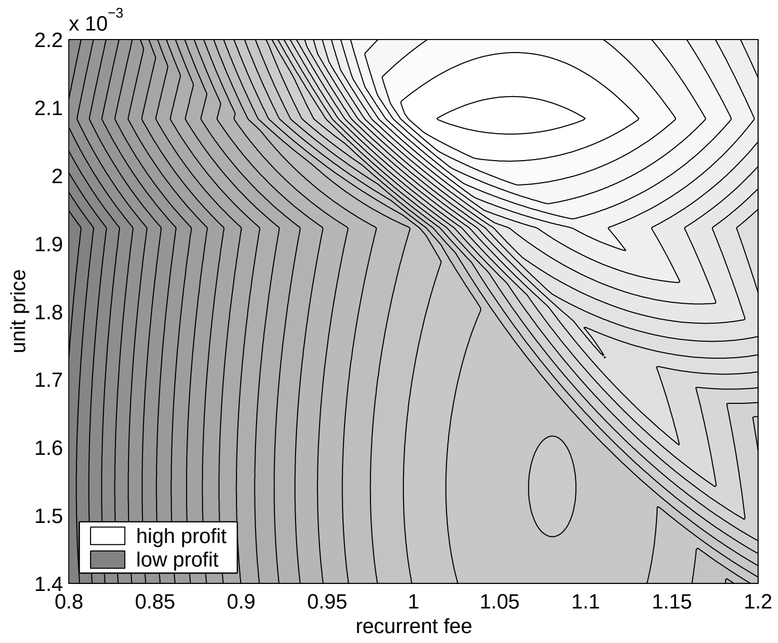


Figure 2.18: Contour plot of profit as function of two decision variables in a market model consisted of one complex product composed of elements subject to flat or utilisation-based pricing.

routine, unlike the previous simulators, does not approximate iteratively the correct output value, there is no reason to expect small fluctuations of $f(\cdot)$. However, local minima or non-differentiability may happen, as shown in Fig. 2.18.[11] Their existence is the result of switching formula (2.15).

One may ask about the correlation between graphs of minimum number of subscribers with that of the profit. This correlation depends on the factors profit is based most on. Logically it has to rely on the number of subscribers since monthly fees are traditionally the most reliable income making factor. However, there may be prices, defined in PM model, for other activities than just for staying with NSP. A customer may be charged for subscription, migration and even for breaking the contract — this, in total, makes more income for NSP than monthly fees only. In such case the income graphs could look entirely different from that in Fig. 2.18. The question is that in case when profit is not based mainly on monthly fees, there is no much sense in specifying limits for q_S. However, while approaching PM models one has to be prepared for presence of both fragmented and non-convex domain as well as for multiple-optima and locally non-differentiable performance index $f(\cdot)$.

[11]This figure illustrates performance index calculated for yet another model that has been presented in [9].

2.4 Conclusions on the existence and nature of difficult problems

The existence of specialised simulators is an irrefutable fact. Models of power systems, waveguides, IP services market introduced in this section are good representatives of a whole branch of simulators dedicated to support good design and operation of complex systems. Simulators presented here share important common features. First, the modelled objects are deterministic, and there is no need to cope with random values present at simulation output.[12] Second, the decision variables are allowed to take values from continuous sets.[13] There exist particularly many simulators that work in a fashion similar to *QW-3D*. They are usually based on FEM which, by spatial and temporal discretisation of physical laws, is able to calculate real-life behaviour of designed systems. Applications of FEM range from ground water flow model to aircraft wing design. The two other simulators considered represent a legion of very specialised, if not to say non-standard, modelling approaches, exploiting one's big knowledge of a particular problem and many-year experience — like in case of IHE and QOSIPS products.

Another feature simulators have common in is they are usually created for manual operation. This means that in interactive mode the operator of a simulator is presented with the current status, or prompted for intermediate decisions where the algorithm is endangered to fail in computing the output value (or the simulation accuracy would be degraded). IHE simulator reports unreachable termination criteria met in its inner loops; *QW-Simulator* creators remind of benefits of setting the discretisation pattern manually. Such actions do not have their counterpart in simulator programming interface (which is mostly a file). This situation limits the possibility of running the simulation-optimisation couple from the very beginning to the end without human intervention. However, this is exactly what customers expect. This is why the optimisation routine must often work in a hostile environment, and must develop workarounds for crisis situations appearing during simulation a skilled operator would be able to resolve or prevent.

Problems presented throughout this chapter show that feasibility constraints are not present widely. They appeared only in the case of IHE models — but it does not mean that such constraints, if met, should disqualify coupling simulator with optimisation routine. First, feasibility constraints emerge as the result of model intricacies: they may define regions where not only the model is invalid, but also they may define states forbidden for the system being modelled, and therefore they are piece of useful information. Sometimes it is only the amount

[12]In the case of market model, the randomness is managed internally.

[13]For market model of the type considered, the prices do not have to be aligned at 0.99 levels.

of work needed that prevents simulator authors from emitting emergency signals in the form understandable by the overlying optimisation routine, before the simulator crashes. Second, all simulators are considered in this work as if they were given to the author by their creators, with no intend (or, sometimes, possibility) of modifying their behaviour. In cases of simulators being commercial products, any modifications were prohibited; in some other cases the amount of work to be put in for those modifications would be prohibitive.

The role implicit constraints may play varies. In some cases (power system model) they determine regions of model validity and no simulation result violating them is reliable. In other cases (market model) they define some objective, but the model is in force also when this objective is not reached. A variation of this type of constraint perception may be when implicit constraints are violated for all $\mathbf{x}$'s in $D_{\mathbf{x}}$, and penalty for their violation alone is the performance function (waveguide model).

As regards the characteristics of performance function $f(\cdot)$, there were observed common and individual features. Common features for all models are the presence of local minima and local nondifferentiability of $f(\cdot)$. The common phenomena for models where output is approximated through iterations are local discontinuities of $f(\cdot)$ and ubiquitous small fluctuations of its value. The awareness of potential problems optimisation that $f(\cdot)$ may cause was common; after all, failure of optimisation routines used originally was the reason of author's involvement in development of new optimisation tools, in cooperation with the developer teams.

The above observations allow to state that there indeed exist simulation-based optimisation problems where constraints defined implicitly are of importance. Problems with feasibility constraints happen rarely but there are reasons that they cannot be reformulated, and therefore they must be handled by appropriate optimisation routines. Implicit constraints are common, although their role may vary. The properties of performance index are problem dependent, but in general they may include discontinuity, multiple minima and simulation error. Those conclusions prove Thesis 1.

The next chapter presents practical problems met by the author in the wider context. Other types of simulation-optimisation problems will be presented along with suitable optimisation algorithms. Such overview helps in formulation of a standard approach to simulation-optimisation, which serves as a guide for solving problems presented in this chapter.

Chapter 3

Survey of problems and commonly used optimisation algorithms

This chapter presents the outcome of literature studies that were made in an attempt put to find practical optimisation problems resembling in a number of ways the problems presented in Chapter 2. The keywords used in the search were ‘simulation’ and ‘optimisation’ (or, rather, in reverse order, since the simulation is, notably peculiar, but just a method of computing the performance index value). In fact, what was looked for was a set of candidate optimisation routines hopefully capable of solving the presented problems. Needless to say, sought routines had eventually to support not keywords but concrete problem features: non-smooth multi-optima performance index and constraints of all presented sorts. The effect of the studies is an anthology of routines that are really useful for the considered problems and problems alike. Also, as a sort of by-product, the effect of literature studies is yet another comprehensive and subjective classification of simulation-optimisation problems; other classifications and overviews within this topic abound in the literature [96, 24, 109].

Unexpectedly enough, majority of publications associate terms ‘simulation’ and ‘optimisation’, when appearing together, with optimisation problems where the dominating difficulty is simulation output obscured by disturbances or noise of random nature. Such cases, apparently different from the three models concerned, are duly presented in this dissertation in Appendix A for two reasons: firstly, because they are so many that they cannot be easily skipped over; and secondly, because some optimisation routines and techniques prescribed by authors in those cases can be of use for the problems concerned in this work.

Such evident shift of gravity towards systems affected by randomness may have its sources in the very definition of the term ‘simulation’. Most authors connote it with imitating state trajectory as the time changes, inevitably with some random input in the game. Consequently, increased simulation effort there means longer simulation horizon. However, simulation, for many having undoubtedly temporal aspect, may also have as its goal increasing accuracy by iterating internal loops in order to balance the model equations, as it is in IHE modeller. Still more groundless is the alleged link between simulation and randomness. Fortunately, there exist authors willing to make a broader definition of simulation [100], easily enclosing the three models of our concern. The cases presented in Chapter 2 are sometimes termed as 'design optimisation' [78, *Optimization Software Guide* link], but such terminology seems characteristic rather for a particular circle of optimisation software manufactures than for a class of problems, and seems markedly unfit for the optimal pricing problem.

The contents of this chapter is as follows. Initially, classification criteria for simulation-optimisation problems are presented. They are followed by some general guidelines for approaching those problems as suggested in the literature, and accompanied by exemplary applications. Next, the main body of the chapter are concise descriptions of most frequently used optimisation routines in this branch. Assuming, as in case of our problems, the performance index type to be the major problem and routine classification criterion, Section 3.1 presents classic, demanding and efficient routines — the gradient ones, and Section 3.2 presents less demanding but more robust — the direct search ones. The selection of algorithms is not only the result of regular literature searches, but it has been affected by experience reported by senior colleagues from — to use tabu search terminology — author's neighbourhood. Many of those routines are constituents of advanced optimisation solvers; the overview of such solvers is given in Section 3.3. The chapter closes with Section 3.4, containing conclusions on the proposed universal approach to solving problems as considered here and alike.

Since most of algorithms presented in this chapter do not support feasibility constraints directly, it is justified to reformulate the optimisation problem (1.1) as follows

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad , \tag{3.1}$$

where $f(\mathbf{x}) \overset{\mathrm{df}}{=} f(\mathbf{x}, \mathbf{s}(\mathbf{x}))$ — that is, to find an optimal point $\mathbf{x}^\star$ that minimises $f(\cdot)$. The set of values the performance index $f(\cdot)$ can take is real and bounded from below.

Simulation-optimisation — features and classifications

Using a simulator to compute an objective function value obviously does not, from strictly theoretical point of view, define by itself a special class of optimisation problems. One can consider employing a simulator to compute the value of a linear function, in the extreme case. However, performing simulation has several practical reasons, preconditions and implications. Usually,

- The performance index or constraints cannot be computed otherwise than numerically; consequently, the optimisation problem becomes nonlinear and, perhaps, discontinuous (cf. models presented in Chapter 2);

- The problem dimension is moderate ($\dim \mathbf{x} < 1000$); this limitation comes partially from restricted capacities of human imagination to circa 7 dimensions, and dates from time when the simulation was started only manually;

- Most of the computation budget is consumed by the simulator, and very little by the optimisation algorithm. This happens due to two major factors. The first is the natural modelling complexity: the load of mathematical and practical knowledge put in a model results in numerous formulas, often appearing in nested loops, solvable by executing iterations of heavy numerics. The second is the uncertainty (e.g. described by probability distribution functions) that propagates within the model rendering it intractable but through lengthy averaging.

Of the above characteristics only the first one really determines efficacy of an optimisation routine being applied. The last one, however, demands efficiency and often performs the ultimate verification of that algorithm applicability.

The following classification criteria of simulation-optimisation problems may be distinguished, in the order of impact they have on selection of the optimisation routine:

1. Type of the decision variables — mixed, discrete only, continuous only.

2. Type of the performance index function — discontinuous, continuous, differentiable, convex, linear.

3. Type of constraints — as for the function type, plus the feasibility constraints.

4. Determinism or uncertainty of the model.

The choice of an optimisation routine is also affected, mainly for efficiency reasons, by a number of other conditions, e.g.:

- Availability of information from simulation other than output;

- Requirement for global optimality of the solution;

- Ability to control simulation behaviour (e.g. accuracy).

These criteria will be shortly presented now.

Continuity of decision variables is presented here as the major classification criterion since it determines validity of the following two others. A variable x_i may be allowed to change continuously or discretely, or both ways. If all decision variables are continuous, we have a continuous optimisation problem; if all decision variables are of the second kind, we have a discrete optimisation problem. All remaining problems are classified as mixed. In reality, most of design problems are formally discrete due to standardisation of available materials, discrete nature of actuators, and alike. Also measurements performed on a system are discrete for the same reasons. However, this fact by itself does not qualify all such problems as discrete — the qualification is determined by the change in system behaviour caused by the smallest possible change of parameter value. If this change is not rapid (i.e. it is rather quantitative than qualitative) then continuous optimisation algorithms may be applied because the discretisation of the algorithm output does not change the nature of a solution. Otherwise (e.g. in case of combinatorial optimisation) one really deals with a discrete problem, and most of the following discussion about continuity-specific features of performance index and constraints does not apply. Discrete problems are not the subject of our concern; however, many of them are attacked using a suite of direct optimisation algorithms, as described in [54], which are useful for our problems. There exist also approaches, like branch-and-bound method, employing continuous algorithms for the discrete optimisation — mainly for their efficiency.

Properties of the function representing performance index greatly determine the solving tactics. If the function exhibits certain desired mathematical properties (linearity, differentiability etc.) then all criteria must be analysed jointly in order to apply a specialised optimisation routine (linear programming can be an example). On the other hand, the more general performance description is the less other criteria are considered (global optimisation through genetic algorithm with constraints handled by penalty function and discreetness handled in genetic operations can be the example).

Constraints on $\mathbf{x}$ are often welcome in optimisation problems as they can significantly reduce the search space, thus accelerating the operation of an algorithm [82, p. 387]. A constraint may result from various reasons: range of control inputs of an object, safety of operation of a modelled system, model validity etc. They are inevitable in design problems; their lack usually means that some part of problem definition process has not been performed carefully enough. All the constraints that are defined explicitly with respect to the decision variables, are desirable, since they can be either supported directly by an optimisation routine (e.g. the linear ones) or handled by appropriate transformations of the design variables [82, p. 383], or by penalty functions.

Determinism or uncertainty of the model is related with the existence of a random model input. Such randomness may be reduced by averaging, thus rendering possible application of all deterministic optimisation routines. Such randomness may also 'get lost' if one is going to use optimisation methods that are themselves of random nature (most global search routines employ randomness that may easily conceal the randomness of the process being optimised). However, the point is to utilise, by the optimisation routine, any knowledge about the random model input. This leads to a whole branch of stochastic optimisation algorithms, presented in Appendix A.

The advantage of querying the simulator for its internal state or extra information would be unquestionable. Every evaluation of $f(\cdot)$ requires running the simulator which is often an opaque piece of software, a black box. It implies that there is no way of getting more information about the simulation outcome than that made available by the software manufacturer. It means particularly that the derivative $\nabla f(\cdot)$ that could speed up optimisation by an order of magnitude, is not available directly. This gradient, as well as higher derivatives, may be estimated by the optimisation routine itself, but since estimation can be inefficient and error-prone, there is strong need for $\nabla f(\cdot)$ to be available directly from the simulator. Gradient availability is postulated particularly often in case of stochastic optimisation. Few specialised simulators — the 'white boxes' [86, p. 19] — offer the feature of gradient computation, hence in most cases the optimisation routine has to estimate it by itself. Standard ways of gradient estimation are presented in Appendix A.

The requirement that the solution found by optimisation routine be globally optimal is virtually not so frequently formulated in practice. Practice shows that a person interested in optimal design done via simulation-optimisation is satisfied rather by a substantial improvement of the performance index than by its global optimality. It is so in the case when the initial solution already exists. Nevertheless, it is always welcome to have an optimisation routine that looks for the global solution, as it may discover a completely unknown attraction area — and may find $\mathbf{x}^\star$ one would never think of. Globality of an optimisation algorithm cannot be obtained without either rigorous assumptions about the performance index and the domain (both being convex, for example), or rather special arrangements concerning the algorithm and making it, usually, not very efficient. The latter case is particularly painful when the simulation times are long. Often the users, unable to make assumptions on $f(\cdot)$, finally resort to evolutionary strategies and other expensive routines [37].

Behaviour of a simulator with some $\mathbf{x}$ on its input can be sometimes controlled by adjusting values of simulation parameters. Those parameters influence such simulation features as running time, overall accuracy, simulator internal rounding and discretisation etc. The ability to accept different parameter values for each simulation run is valuable simulator feature. The question is whether the optimisation routine can make use of it. If so, rough simulation can be done in a preliminary stage of optimisation, followed then by simulating with finer and still finer accuracy as the solution is being approached. Formally, those general simulation parameters can be perceived as extra elements of the vector $\mathbf{x}$ of decision variables.

Simulation-optimisation — guidelines for solving

When faced with simulation-optimisation problems similar to those presented in Chapter 2 (i.e. characterised by multioptima discontinuous performance index, troublesome constraints, costly performance index calculations), one is advised by numerous authors — reasonably enough — to put most effort in the underlying model examination and recognition. Having gained the necessary knowledge, one may apply the following techniques to approach the problem:

- Verify necessity and formulation of original constraints; reformulate the constraints;
- Replace the original model, where possible, with its rougher or local counterpart;
- Develop specialised optimisation routines fit to handle the particular problem.

Those techniques will be presented below shortly.

Verifying constraints. In the literature there can be found many guidelines for optimisation problem preparation. As the constraints are regarded, most authors advise to take a closer look at the model first [82, pp. 382–399]. The constraints can be classified as natural (implied by natural laws of the modelled system) or practical (resulting from common sense, former practice, and placed to accelerate numerical searches). Initially, only natural constraints should be considered and, if possible, eliminated through appropriate transformations and introduction of slack variables — so that the nature of optima does not change. Any implicit constraint whose nature can be identified, should be mapped onto search domain [11, p. 189]. Finally, only those practical constraints are added that are desirable either to maintain model validity or to speed up the computations. One has to be particularly careful while applying the practical constraints — search space reduction may really accelerate working of the algorithm, but when applied over-eagerly, it can deprive the domain of the optimal solution altogether.

If the above procedure had been followed and implicit constraints were still active, then the application of a penalty function could have been the solution. Various penalising schemes are discussed [96, pp. 487–517]. The penalty function can, however, be applied only when one knows the extent to which the constraints are violated. If even those few data are unavailable, e.g. due to simulation failure (as presented in Section 2.1), then the only approach found in textbooks [116, pp. 138–143] suggests to set the penalty function value to an infinite value. Unfortunately, by so doing one makes the response surface like a sieve, with which few solvers can cope.

Using an alternative model. Like in case of stochastic approximation (cf. Section A.1), there is much emphasis put on model reformulation in order to either yield from it more information (mostly, the gradient) or to speed up calculation, or both. In very many publications (see [50, pp. 1–9] and bibliography cited in the preceding paragraph) the authors insist on taking a deeper look into the model, and overcoming the difficulties by some kind of aggregation, mapping, linearisation and heuristics. If such alternative models are to be linear, their construction and use happens usually in the final stage of optimisation run. If such models are nonlinear, they are made to provide some estimate of $f(\cdot)$ (analogously to branch-and-bound approach) in the initial optimisation phase, to indicate unpromising regions.

The evidence of practical applications of simplified models is strong. For example, analytical models have been successfully constructed and used in problem of design of an impedance transformer [8]. Also, another successful attempt was made very recently [64] to make polynomial and theory-based approximation of frequency characteristics defining the performance index in the case of microwave filter design problem. In this case, the model simplification was the result of much human effort and deep knowledge — and it allowed to apply SQP optimisation routine (see p. 67).

In many cases, the original models are used once or when needed to determine or adjust parameters of the simpler model. Also frequently the goal of doing so is only to accelerate output calculation, because the problem is going to be attacked with nongradient routines anyhow. As an example, an automatic design procedure of a micropump is reported in [70] where the costly FEM simulation was applied only to find the coefficients of some much faster mathematical model (computed by *PSPICE* circuit simulator), which was then explored by a genetic optimisation algorithm. In another example [47] a neural network is used to approximate the original model output in order to make predictions of the performance index, thus avoiding costly computation of the original model at possibly bad design points. The approach was applied for optimal design of a job shop system.

Developing tailored routines. Tailoring an optimisation routine to make it solve a particular task rarely consists in changing the routine structure, e.g. switching statements. Usually, routine adaptation to the problem means literally adaptation of some parameters (distributions, coefficients, trial point pool sizes etc.) or construction of appropriate structures and operators[1] (neighbourhoods, random selection, offspring generation etc.). Of course, proper routine adaptation may require as much original model knowledge as in case of alternative model making. Another ‘tailoring’ may take place when one is to apply a hybrid optimisation algorithm consisting of two or more standard optimisation routines. In such case, the problem is where to cut, that is, when to stop one routine and switch to another, and what should the termination criterion be. The problem of where to cut seems to be actually more pronounced than the problem of termination criterion for the algorithm as a whole, because as a whole an algorithm can be run until the last moment when the solution is requested.

Two interesting cases of successful merging of optimisation routines into hybrids can be given here where the hybrids are made not as general-purpose but as problem-specific algorithms. The first one [21] uses a histogram of objective values, created by some standard direct search algorithm, as starting point for problem-specific routine that performs classification of decision variables and variable values as ‘important/unimportant’ and ‘good/bad’. Then the solution is picked that satisfies problem specification. Genetic algorithm, simulated annealing and tabu search are the considered direct search routines. The second case [47], already cited, is a merger of genetic algorithm and tabu search features — the authors present a technique called scatter search, similar to genetic algorithm, with the difference that the offspring is created deterministically by linear combinations of so-called reference points, i.e. good solutions obtained in the previous algorithm steps.

3.1 Gradient methods

Gradient optimisation algorithms use in each step information about first or second order derivative of performance index $f(\cdot)$. Optimisation is performed subject to equality or inequality constraints specified by functions

$$\text{a)} \left\{ \begin{array}{lcl} h_{\mathrm{I},1}(\mathbf{x}) & \leq & 0 \\ & \vdots & \\ h_{\mathrm{I},N_\mathrm{I}}(\mathbf{x}) & \leq & 0 \end{array} \right. , \qquad \text{b)} \left\{ \begin{array}{lcl} h_{\mathrm{E},1}(\mathbf{x}) & = & 0 \\ & \vdots & \\ h_{\mathrm{E},N_\mathrm{E}}(\mathbf{x}) & = & 0 \end{array} \right. , \tag{3.2}$$

that can be expressed shortly $\mathbf{h}_\mathrm{E}(\cdot) = \mathbf{0}$ and $\mathbf{h}_\mathrm{I}(\cdot) \leq \mathbf{0}$. Gradient methods are deterministic and local, and therefore suitable for optimisation with models that have undergone serious adaptations. However, the effort put in fitting a problem to an algorithm pays off in the performance superior to that of direct search methods, especially for moderate and large dimension problems.

[1]Skillful construction of structures and operators may also be a means to get rid of difficult constraints unsupportable by the optimisation routine directly.

Three optimisation routines have been selected to be presented here; they are SLP, SQP and GRG. Their selection was driven by wide scope of problems they cover, and by their mere popularity resulting from efficiency and efficacy. They all operate utilising in each iteration an improving direction $\mathbf{d}$, but the way $\mathbf{d}$ is made diversifies algorithm properties.

SLP — Sequential linear programming

Sequential linear programming routine (see e.g. [16, pp. 432–437]) uses first-order approximate of performance index $f(\cdot)$ and constraints (3.2), both assumed to be continuously differentiable. In each iteration $f(\cdot)$ and constraints are linearised at the current solution approximate $\mathbf{x}_k$, and the resulting linear programming (LP) problem is solved subject to additional constraints defining the trust region. The trust region represents limited range of such linear approximations; in particular it guards against eventual linearised problem unboundedness that may appear. Depending on the quality of the solution found by LP (i.e. on performance index improvement and constraints satisfaction), it may be accepted as the new solution approximate, or rejected. Successive LP problem formulations are affected by trust region adaptations. In practical applications SLP works with the performance index augmented with penalty factors

$$\tilde{f}(\mathbf{x}) = f(\mathbf{x}) + \mu \left[\sum_{i=1}^{N_{\mathrm{I}}} \max\{0, h_{\mathrm{I},i}(\mathbf{x})\} + \sum_{i=1}^{N_{\mathrm{E}}} |h_{\mathrm{E},i}(\mathbf{x})| \right] \quad , \tag{3.3}$$

with μ being some penalty parameter. The outline of SLP algorithm is presented below:

Step 0: Initialise with a feasible start point $\mathbf{x}_0$ and with hypercube trust region around, of size determined by vector $\boldsymbol{\delta}$.

Step 1: Solve LP with (3.2) and (3.3) linearised around $\mathbf{x}_k$

$$\begin{array}{rl} \min_{\mathbf{d},\mathbf{a}_{\mathrm{I}},\mathbf{a}_{\mathrm{E}}^{+},\mathbf{a}_{\mathrm{E}}^{-}} & f(\mathbf{x}_k) + [\nabla f(\mathbf{x}_k)]^T \mathbf{d} + \mu \left[\sum_{i=1}^{N_{\mathrm{I}}} a_{\mathrm{I},i} + \sum_{i=1}^{N_{\mathrm{E}}} (a_{\mathrm{E},i}^{+} + a_{\mathrm{E},i}^{-}) \right] \\ \text{subject to} & \mathbf{a}_{\mathrm{I}} \geq \mathbf{h}_{\mathrm{I}}(\mathbf{x}_k) + \nabla \mathbf{h}_{\mathrm{I}}(\mathbf{x}_k)\mathbf{d} \quad , \\ & (\mathbf{a}_{\mathrm{E}}^{+} - \mathbf{a}_{\mathrm{E}}^{-}) = \mathbf{h}_{\mathrm{E}}(\mathbf{x}_k) + \nabla \mathbf{h}_{\mathrm{E}}(\mathbf{x}_k)\mathbf{d} \quad , \\ & -\boldsymbol{\delta} \leq \mathbf{d} \leq \boldsymbol{\delta} \quad , \\ & \mathbf{a}_{\mathrm{I}}, \mathbf{a}_{\mathrm{E}}^{+}, \mathbf{a}_{\mathrm{E}}^{-} \geq \mathbf{0} \quad , \end{array} \tag{3.4}$$

with $\mathbf{a}_{\mathrm{I}}$, $\mathbf{a}_{\mathrm{E}}^{+}$ and $\mathbf{a}_{\mathrm{E}}^{-}$ being slack variables.

Step 2: Terminate if $\mathbf{d} = \mathbf{0}$. Otherwise calculate approximation accuracy — this is based on comparisons of (3.3) vs. objective function in (3.4) — and if it is not satisfactory then decrease $\boldsymbol{\delta}$ and go to Step 1.

Step 3: Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}$. Perform further adjustments of $\boldsymbol{\delta}$ and go to Step 1.

SLP is reported to be efficient in highly constrained, large and non-linear optimisation problems. However, if the solution happens not to lie on one of feasible region vertices, the algorithm performance is much degraded. A solution that might be applied in such case is to accept $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha\mathbf{d}$, with $\alpha \in (0, 1)$ being another adjustable algorithm parameter. A routine so modified, called sequential linearisation with relaxation (SLR), worked well in case of optimal price setting problem — the protoplast of optimal pricing problem presented in Section 2.3.

SQP — Sequential quadratic programming

To overcome SLP drawbacks, the SQP routine (see e.g. [18, pp. 55–84]) goes a step further and utilises information about the second order derivative of $f(\cdot)$. The problem formulation is as in (3.1) and in (3.2), with the assumption that all functions are continuously twice-differentiable. In each iteration of SQP algorithm a quadratic programming (QP) routine finds minimum of quadratic approximation of the original problem Lagrangian function

$$L(\mathbf{x}, \boldsymbol{\lambda}_\mathrm{I}, \boldsymbol{\lambda}_\mathrm{E}) = f(\mathbf{x}) + \boldsymbol{\lambda}_\mathrm{I}^T \mathbf{h}_\mathrm{I}(\mathbf{x}) + \boldsymbol{\lambda}_\mathrm{E}^T \mathbf{h}_\mathrm{E}(\mathbf{x}) \quad , \tag{3.5}$$

with $\boldsymbol{\lambda}_\mathrm{I}$ and $\boldsymbol{\lambda}_\mathrm{E}$ being the Lagrange multipliers. Therefore, the function being optimised by QP is not merely an appropriate approximation of $f(\cdot)$ but also carries information about constraints curvature. QP sub-problem solution is used to construct the direction for line optimisation of so-called merit function, defined exactly as in (3.3). SQP algorithm can be presented as follows:

STEP 0: Initialise with $\mathbf{x}_0$, $\boldsymbol{\lambda}_{\mathrm{I},0}$, $\boldsymbol{\lambda}_{\mathrm{E},0}$ and with some initial and positively definite approximate $\mathbf{H}_0$ of Hessian $\nabla^2_{\mathbf{xx}} L(\mathbf{x}_0, \boldsymbol{\lambda}_{\mathrm{I},0}, \boldsymbol{\lambda}_{\mathrm{E},0})$. Usually, $\mathbf{H}_0 = \mathbf{I}$.

STEP 1: Solve QP problem for second order approximate of (3.5) and for first order approximate of (3.2) at $\mathbf{x}_k$, $\boldsymbol{\lambda}_{\mathrm{I},k}$, $\boldsymbol{\lambda}_{\mathrm{E},k}$

$$\begin{aligned} \min_{\mathbf{d}} \quad & f(\mathbf{x}_k) + [\nabla f(\mathbf{x}_k)]^T \mathbf{d} + \tfrac{1}{2}\mathbf{d}^T \mathbf{H}_k \mathbf{d} \\ \text{subject to} \quad & \mathbf{h}_\mathrm{I}(\mathbf{x}_k) + \nabla \mathbf{h}_\mathrm{I}(\mathbf{x}_k)\mathbf{d} \le 0 \quad , \\ & \mathbf{h}_\mathrm{E}(\mathbf{x}_k) + \nabla \mathbf{h}_\mathrm{E}(\mathbf{x}_k)\mathbf{d} = 0 \quad , \end{aligned} \tag{3.6}$$

By-products of QP solving are Lagrange multipliers $\tilde{\boldsymbol{\lambda}}_\mathrm{I}$ and $\tilde{\boldsymbol{\lambda}}_\mathrm{E}$ at the solution point $\mathbf{d}$.

STEP 2: Perform line search, $\min_{\alpha \in <0,1>} \tilde{f}(\mathbf{x}_k + \alpha\mathbf{d})$, with $\tilde{f}(\cdot)$ as in (3.3). Terminate if no significant improvement in $f(\cdot)$ or no significant solution shift α was made, and if simultaneously the constraints are satisfied or violated negligibly.

STEP 3: Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha\mathbf{d}$. Adopt $\tilde{\boldsymbol{\lambda}}_\mathrm{I}$ and $\tilde{\boldsymbol{\lambda}}_\mathrm{E}$ as $\boldsymbol{\lambda}_{\mathrm{I},k+1}$ and $\boldsymbol{\lambda}_{\mathrm{E},k+1}$, respectively. Use $\mathbf{H}_k$ together with gradients of $L(\cdot)$ calculated at $\mathbf{x}_k$, $\boldsymbol{\lambda}_{\mathrm{I},k}$, $\boldsymbol{\lambda}_{\mathrm{E},k}$ and at $\mathbf{x}_{k+1}$, $\boldsymbol{\lambda}_{\mathrm{I},k+1}$, $\boldsymbol{\lambda}_{\mathrm{E},k+1}$ to make the current estimate $\mathbf{H}_{k+1}$, using e.g. BFGS formula. Go to STEP 1.

Refined algorithm organisation makes SQP outperform SLP: no zigzagging of $\{\mathbf{x}_k\}$ is observed due to more complete approximation of $f(\cdot)$ coming at low cost thanks to BFGS updates of $\mathbf{H}_k$. However, SQP may be vulnerable to problems where quadratic $f(\cdot)$ approximation is inaccurate (the result may be QP's with conflicting constraints) or where the constraints are highly non-linear. Consequently, proposals of algorithm improvements appear, e.g. concerning keeping $\{\mathbf{x}_k\}$ within the feasible region. SQP efficiency is influenced, in terms of the problem properties, mostly by the number of inequality constraints; and in terms of implementation, by the quality of the underlying QP solver. However, the main precondition for proper SQP operation is that it should be run in the proximity of $\mathbf{x}^\star$. SQP is a recognised and appreciated advanced routine; its many applications span multiple domains. From our point of view an important one was to employ SQP in the process of surface filter design [64]. It is, however, the profound reformulation of the used model that constitutes the major part of the work reported there and that makes SQP application possible. The routine implementation that was used was an off-the-shelf one, by Mathworks [68].

GRG — Generalised reduced gradient

In GRG the distinction is made between the performance index arguments that are decision variables, and the arguments that are dependent variables. Thus, the original problem definition formulated in (1.1, 1.3) is restored.[2] The optimisation is performed subject to constraints (1.4, 1.5). Subsequent approximations of $\mathbf{x}^\star$ are generated by GRG routine — similarly to SQP — through line searches, but the way search directions are constructed is entirely different. Consequently, the routine properties are different too — although both GRG and SQP find local solutions for general nonlinear and differentiable optimisation problems. In GRG (see e.g. [36]) the key term that search direction construction bases on is the general reduced gradient of $f(\cdot)$, i.e. a gradient reduced to the subspace of decision variables $\mathbf{x}$. The value of generalised reduced gradient has two components; one is the performance index gradient projected directly on the subspace of $\mathbf{x}$, another is the performance index gradient projected on the subspace of $\mathbf{y}$ and mapped subsequently (by means of linearised equality constraints) onto $\mathbf{x}$. Search vector based on so computed gradient is constructed in every algorithm iteration,

[2]The support for non-linear inequality constraints, like those present explicitly in SLP and SQP descriptions, can be accomplished for GRG by introduction of slack variables.

and line search procedure is executed in order to find new solution approximation. If this solution happens to be infeasible, its feasibility is restored by projection onto the subspace of $\mathbf{x}$. GRG algorithm may be put down as follows:

STEP 0: Initialise with a feasible start solution approximation $\mathbf{x}_0$, $\mathbf{y}_0$.

STEP 1: Compute the reduced gradient value at $\mathbf{x}_k$, $\mathbf{y}_k$

$$\tilde{\mathbf{g}}_k^T = [\nabla_{\mathbf{x}} f(\mathbf{x}_k, \mathbf{y}_k)]^T - [\nabla_{\mathbf{y}} f(\mathbf{x}_k, \mathbf{y}_k)]^T [\nabla_{\mathbf{y}} \mathbf{h}(\mathbf{x}_k, \mathbf{y}_k)]^{-1} \nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x}_k, \mathbf{y}_k) \ . \tag{3.7}$$

(It is assumed here that (1.3) is such that $\nabla_{\mathbf{y}} \mathbf{h}(\cdot)$ is square and non-singular.) Next, prepare components of search direction $\mathbf{d}_{\mathbf{x},k}$ in the space of $\mathbf{x}$

$$d_{\mathbf{x},k,i} = \begin{cases} 0 & \text{if } \tilde{g}_{k,i} < 0 \text{ and } x_{k,i} = x_{\mathrm{U},i} \\ 0 & \text{if } \tilde{g}_{k,i} > 0 \text{ and } x_{k,i} = x_{\mathrm{L},i} \\ -\tilde{g}_{k,i} & \text{otherwise} \ . \end{cases} \tag{3.8}$$

STEP 2: Prepare components of search direction $\mathbf{d}_{\mathbf{y},k}$, in the space of $\mathbf{y}$; the search direction is to be tangent to equality constraints

$$\mathbf{d}_{\mathbf{y},k} = - [\nabla_{\mathbf{y}} \mathbf{h}(\mathbf{x}_k, \mathbf{y}_k)]^{-1} \nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x}_k, \mathbf{y}_k) \mathbf{d}_{\mathbf{x},k} \ . \tag{3.9}$$

STEP 3: Perform line search: $\alpha_k = \arg\min f(\mathbf{x}_k + \alpha_k \mathbf{d}_{\mathbf{x},k}, \mathbf{y}_k + \alpha_k \mathbf{d}_{\mathbf{y},k})$ subject to constraints (1.5). Set $\mathbf{x}_{k+1} = \mathbf{x} + \alpha_k \mathbf{d}_{\mathbf{x},k}$.

STEP 4: Terminate if line search has found a solution lying too close to the previous one in the space of $\mathbf{x}$, i.e. if $|\alpha_k d_{\mathbf{x},k,i}| < \epsilon_i$ for all i's.

STEP 5: Solve (1.3) for $\mathbf{y}$ and set $\mathbf{y}_{k+1}$ to be equal to that solution. (Such procedure discards the new solution candidate in the space of $\mathbf{y}$ found in STEP 2, but that solution is usually infeasible. This workaround restores solution feasibility.) Go to STEP 1.

GRG suits well problems with relatively few decision variables as compared to the number of dependent variables. Unlike SQP, it sticks closely to the feasible region, and is therefore recommended in cases of strongly nonlinear $\mathbf{h}(\cdot)$'s. Unlike SQP, it does not rely on second order derivatives, which results as well in degraded performance for convex problems as in generally improved robustness.

One of many successful GRG applications worth citing [66] is a case where the optimal working point of gas network was to be found. However, it turned out there that the routine in the form given above was far too immature to be useful in practice, even after numerous upgrades suggested in the literature had been incorporated in the algorithm. Only after the author had profoundly analysed optimisation results and introduced her own improvements, algorithm undesired behaviours (like zigzagging and slow convergence) were tamed. One of important conclusions conveyed there was that commercial implementations of optimisation routines have substantial added value, and tips and tricks for their effective working are kept secret by manufacturers.

3.2 Direct search methods

The algorithms discussed in this section are directly value-based, i.e. their behaviour is determined only by the value of the performance index, without any support of the performance derivatives. On one hand, the absence of such support must definitely have an adverse impact on algorithm efficiency, especially for medium and large scale problems. On the other, it improves the algorithm robustness. This tradeoff is different for every algorithm. This is because different assumptions are made about $f(\cdot)$, and therefore various direct search methods may be efficient at the cost of robustness, and vice versa. Another important direct search methods feature that wins them many eager supporters is that, giving up altogether the idea of gradients, they could be applied for combinatorial optimisation.

The evident drawback of direct search methods is their rapidly decreasing efficiency with the growth of problem dimension. Fortunately, in simulation-optimisation problems similar to ours the number of decision variables is moderate or it can be made moderate by some sort of aggregation. To start with, one may consider applying direct search in its possibly the purest and simplest imaginable form: the random search technique, which is choosing at random a trial point from $D_{\mathbf{x}}$ and evaluating $f(\cdot)$ there until a satisfactory solution is found. Fortunately, very few optimisation problems require resorting to such a brute scheme.

Powell algorithm

There exist a couple of routines stemming directly from Newton methods, and following the idea of subsequent directional minimisations. Once the starting point $\mathbf{x}_0$ is given, there is always the dilemma how to generate directions for the line search subroutine. There are several approaches: Gauss-Seidel, Rosenbrock, Hooke-Jeeves [41, pp. 97–99 and pp. 106–109]. In yet another one of them, the Powell routine [41, pp. 113–123], the Hessian matrix approximation, used in next line search direction preparation, is made using solely the data from the last $\dim \mathbf{x}$ searches. The routine maintains quadratic convergence but its drawback is that search directions tend to become linearly dependent, which has to be cancelled by periodic algorithm restarts. The algorithm for a problem of size N is as follows:

STEP 0: Let $\mathbf{x}_0$ be a given starting point.

STEP 1: Create the set of directions $R = \{\mathbf{d}_1, \ldots, \mathbf{d}_N\}$ and initialise its elements to search space versors, $\mathbf{d}_i := \mathbf{e}_i$.

STEP 2: Perform N line searches; start a single line search $i = 1, \ldots, N$ from the point $\mathbf{x}_{i-1}$ along $\mathbf{d}_i$ and call the result $\mathbf{x}_i$. Stop either if the maximum number evaluations of $f(\cdot)$ has been reached or if $f(\mathbf{x}_i) > (1-\epsilon)f(\mathbf{x}_{i-1})$, ϵ being the relative performance index value improvement.

STEP 3: Remove $\mathbf{d}_1$ from R and shift the remaining directions, $\mathbf{d}_i := \mathbf{d}_{i+1}$. Complete R with $\mathbf{d}_N := \mathbf{x}_N - \mathbf{x}_0$.

STEP 4: Perform an additional line search from $\mathbf{x}_N$ along $\mathbf{d}_N$ and call the result $\mathbf{x}_0$. Stop if the termination criterion (like in STEP 2) is satisfied; else return to STEP 1.

This routine does not require exact line searches. A choice of line search algorithms and practical suggestions for the implementation can be found in [90, pp. 397–408]. Powell method acquires quadratic approximation of $f(\cdot)$ with all its consequences: suboptimality of the solution and vulnerability to performance index singularities, although mitigated somewhat by periodic method restarts.

Nelder-Mead simplex search

Nelder-Mead simplex search routine starts here a suite of various heuristic optimisation approaches. The workings of heuristic methods, like Nelder-Mead simplex search, do not have such strict theoretical foundations and assumptions (e.g. Taylor series approximation, Karush-Kuhn-Tucker (KKT) optimality conditions) as the gradient routines, of which some have been presented above. However, heuristic routines showed to be very efficient in practice, despite the lack of appropriate convergence proofs. At the cost of much degraded performance in regular optimisation problems, they help where deterministic routines fail. Also, easiness of their implementation cannot be neglected as it encourages modifications of the original algorithm that can be done even by practitioners with weak mathematical background.

Nelder-Mead routine [77] maintains in each its step a simplex, i.e. a set of $N + 1$ points, N being the problem dimension. The points constituting the initial simplex vertices may be picked at random or may be provided from outside. The simplex is potentially transformed in subsequent iterations by reflection, expansion, contraction and multiple contraction operations

— each of them yields a new trial point (or points) which, hopefully, can replace bad vertices. Therefore, each iteration starts with an attempt to reflect the worst (in terms of $f(\cdot)$ value) vertice. If quality of so obtained trial point is promising, another trial is made to expand the simplex farther in that reflection direction. On the contrary, if no progress is detected, a contraction is made: the unpromising vertice collapses towards the simplex centre. Finally, if the contraction operation does not help to find any better point to replace that worst vertice, a multiple contraction towards the best simplex node is executed for all other simplex vertices. In optimum vicinity multiple contractions make the simplex small (in terms of its radius) and the termination criterion is finally passed. Therefore, the algorithms goes as follows:

STEP 0: Given a set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_{N+1}\}$ of simplex vertices, evaluate $f(\cdot)$ at each of them.

STEP 1: Find vertice $\mathbf{x}_h$ such that $\forall_{i=1,\ldots,N+1} f(\mathbf{x}_h) \geq f(\mathbf{x}_i)$, and vertice $\mathbf{x}_l$ such that $\forall_{i=1,\ldots,N+1} f(\mathbf{x}_l) \leq f(\mathbf{x}_i)$. Calculate centre $\mathbf{c}$ of the current simplex X deprived of the worst point, $\mathbf{c} = \frac{1}{N}\left(\mathbf{x}_1 + \cdots + \mathbf{x}_{h-1} + \mathbf{x}_{h+1} + \cdots + \mathbf{x}_{N+1}\right)$. Stop if $f(\mathbf{x}_h) > (1-\epsilon) f(\mathbf{x}_l)$.

STEP 2: Reflection. Make a trial point $\tilde{\mathbf{x}}_1 = (1+\alpha)\mathbf{c} - \alpha \mathbf{x}_h$, with $\alpha > 0$ being the reflection coefficient. If $\forall_{i=1,\ldots,N+1,\ i\neq h} f(\tilde{\mathbf{x}}_1) > f(\mathbf{x}_i)$ then go to STEP 4, else replace $\mathbf{x}_h$ in X with $\tilde{\mathbf{x}}_1$ and go to STEP 3

STEP 3: Expansion. If $f(\tilde{\mathbf{x}}_1) > f(\mathbf{x}_l)$ then go to STEP 1. Otherwise make another trial point $\tilde{\mathbf{x}}_2 = -\gamma \mathbf{c} + (1+\gamma)\tilde{\mathbf{x}}_1$, with $\gamma > 1$ being the expansion coefficient. If $f(\tilde{\mathbf{x}}_2) < f(\mathbf{x}_l)$ then replace $\tilde{\mathbf{x}}_1$ in X with $\tilde{\mathbf{x}}_2$. Go to STEP 1.

STEP 4: Contraction. If $f(\tilde{\mathbf{x}}_1) < f(\mathbf{x}_h)$ then replace $\mathbf{x}_h$ in X with $\tilde{\mathbf{x}}_1$ and give $\tilde{\mathbf{x}}_1$ the name $\mathbf{x}_h$. Make another trial point $\tilde{\mathbf{x}}_3 = (1-\beta)\mathbf{c} + \beta \mathbf{x}_h$, with $0 < \beta < 1$ being the contraction coefficient. If $f(\tilde{\mathbf{x}}_3) < \mathbf{x}_h$ than replace $\mathbf{x}_h$ in X with $\tilde{\mathbf{x}}_3$ and go to STEP 1; else go to STEP 5.

STEP 5: Multiple contraction. Replace every $\mathbf{x}_i$ in X with $\frac{1}{2}(\mathbf{x}_l + \mathbf{x}_i)$. Go to STEP 1.

The method is more robust than Powell algorithm but in some cases the simplex tends to ‘flatten’ itself against steep slopes of $f(\cdot)$, requiring a form of restart — like Powell algorithm — e.g. by replacing one vertice with a randomly chosen point. Support of constraints is usually accomplished through penalty functions. Stop criterion given in STEP 1 is only one of many possible; other popular ones are the maximum number of $f(\cdot)$ evaluations, absolute accuracy, a certain value of $f(\cdot)$ reached, simplex radius etc.

Simulated annealing

Simulated annealing routine was first proposed in [71] for stochastic optimisation — but soon it was used also for deterministic optimisation problems. Unlike simplex search, the routine works with just one point only in each iteration — that is, with current solution approximation $\mathbf{x}_n$. It makes attempts to improve it in step $n+1$ by choosing at random some candidate point $\tilde{\mathbf{x}}_{n+1}$ in the neighbourhood of $\mathbf{x}_n$, and accepting it according to the following formula

$$\mathbf{x}_{n+1} = \begin{cases} \tilde{\mathbf{x}}_{n+1} & \text{if } f(\tilde{\mathbf{x}}_{n+1}) < f(\mathbf{x}_n) \ , \\ \tilde{\mathbf{x}}_{n+1} & \text{with probability } \ \exp\left(-\frac{f(\tilde{\mathbf{x}}_{n+1})-f(\mathbf{x}_n)}{\theta_n}\right) \ \text{ if } f(\tilde{\mathbf{x}}_{n+1}) > f(\mathbf{x}_n) \ , \\ \mathbf{x}_n & \text{with probability } \ 1-\exp\left(-\frac{f(\tilde{\mathbf{x}}_{n+1})-f(\mathbf{x}_n)}{\theta_n}\right) \ \text{ if } f(\tilde{\mathbf{x}}_{n+1}) > f(\mathbf{x}_n) \ , \end{cases} \tag{3.10}$$

where θ_n is the current 'temperature', i.e. a coefficient allowing the algorithm to climb uphill in search for global optimum. θ_n must decrease to zero as the algorithm proceeds. The routine extreme simplicity makes it applicable for very wide range of problems, including stochastic and discrete ones.

CRS — Controlled random search

CRS routine, presented in its original form in [91], is similar to Nelder-Mead simplex search in that it maintains a pool of points and produces new trial points by reflections of the worst point in the pool. However, CRS pool is much bigger than $\dim \mathbf{x} + 1$, and the reflection centre is calculated for a small subset of points randomly chosen from it. This feature gives CRS globality flavour that neither Powell nor Nelder-Mead have. The original CRS algorithm (called here CRS1 since it was the starting point for many later modifications) is as follows:

Step 0: Make the initial pool $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of distinct points, where $N \gg \dim \mathbf{x}$ by picking them at random from the optimisation domain. Evaluate $f(\mathbf{x}_i)$ for every $i = 1, \ldots, N$.

Step 1: Find in the current pool X a point $\mathbf{x}_h$ with worst performance index, and $\mathbf{x}_l$ with best performance index there. Stop if some termination criterion (e.g. $|f(\mathbf{x}_h) - f(\mathbf{x}_l)| < \epsilon$) is satisfied.

Step 2: Choose at random $\dim \mathbf{x} + 1$ distinct points from pool X that will constitute a simplex $\tilde{X} = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_{\dim \mathbf{x}+1}\}$, i.e. a subset of X.

Step 3: Calculate a trial point $\tilde{\mathbf{x}} = 2\mathbf{c} - \tilde{\mathbf{x}}_{\dim \mathbf{x}+1}$ by reflecting $\tilde{\mathbf{x}}_{\dim \mathbf{x}+1}$ through centre $\mathbf{c} = \frac{1}{\dim \mathbf{x}} \sum_{i=1}^{\dim \mathbf{x}} \tilde{\mathbf{x}}_i$.

Step 4: If $\tilde{\mathbf{x}}$ is in the search domain, then evaluate $f(\tilde{\mathbf{x}})$, else go to Step 2.

Step 5: If $f(\tilde{\mathbf{x}})$ is worse than $f(\mathbf{x}_h)$ go to Step 2.

Step 6: Replace $\mathbf{x}_h$ in X by $\tilde{\mathbf{x}}$ (i.e. set $X := \{\tilde{\mathbf{x}}\} \cup X \backslash \mathbf{x}_h$) and go to Step 2.

This basic scheme, CRS1, was followed by numerous modifications (nicely described in [1]), made to improve efficiency. In the first modification, CRS2, particularly widely cited, the simplex is not chosen completely randomly but is forced to contain $\mathbf{x}_l$:

Step 2: Make a simplex $\tilde{X} = \{\mathbf{x}_l\}$. Enlarge simplex with $\dim \mathbf{x}$ distinct points chosen at random from pool X.

Such change accelerates convergence, which is considered the weakest point of CRS, especially if the solution lies on constraints (consider that the only means of handling $\tilde{\mathbf{x}}$ are to accept it or to reject it altogether; no e.g. projection is envisaged).

Further modifications of CRS aim to improve convergence by activating occasionally some local algorithm, be it the very Nelder-Mead search (in CRS3), intensified sampling around $\tilde{\mathbf{x}}$ performed when $f(\tilde{\mathbf{x}}) < f(\mathbf{x}_l)$ (CRS4), gradient-based search (CRS5) or quadratic interpolation of $f(\cdot)$ (CRSI). From the point of view of this dissertation it may be significant to observe that local routines are activated relatively easily, e.g. already if $\tilde{\mathbf{x}}$ is better than $\frac{N}{10}$-th best point in X [92], and that some CRS's switch perpetually between local and global (original CRS) routines.

Like in Nelder-Mead, support for implicit constraints is usually accomplished through penalty functions. (It could also be effectuated in way usual to CRS, by discarding $\tilde{\mathbf{x}}$, but this is not done since no one knows how seriously the implicit constraints reduce the chance of producing a feasible $\tilde{\mathbf{x}}$.)

COMPLEX — Constrained simplex search

COMPLEX routine [20] is an important upgrade of Nelder-Mead simplex search that has a built-in support for explicit and implicit constraints provided that the constrained domain is convex. It works by manipulating in each iteration a pool X of points (also referred to as a complex of points), where $\bar{\bar{X}} \geq \dim \mathbf{x} + 1$. Bigger number of points than in Nelder-Mead routine helps in maintaining complex regularity — but it is not as big and costly as for CRS. The point pool is created by a series of augmentation operations performed around some initial feasible point $\mathbf{x}_0$. In each iteration the worst point in the pool is reflected — and its image is moved back towards the reflection centre until it outperforms the worst point. Handling of violated implicit constraints is accomplished through the same mechanism of backtracking towards the reflection centre. COMPLEX algorithm can then be written as follows:

Step 0: Given an initial feasible point $\mathbf{x}_0$, create a pool $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$, $k \geq \dim \mathbf{x}+1$ by a series of augmentation operations. Initially, $X = \{\mathbf{x}_0\}$. After the augmentation, evaluate $f(\cdot)$ at every element in X.

Augmentation. Calculate centre $\mathbf{c}$ of X. Choose at random a trial point $\tilde{\mathbf{x}}$ such that explicit constraints (1.4a) are satisfied. If $\tilde{\mathbf{x}}$ satisfies also implicit constraints (1.4b), include $\tilde{\mathbf{x}}$ into the pool, $X := X \cup \{\tilde{\mathbf{x}}\}$. Otherwise move $\tilde{\mathbf{x}}$ gradually towards complex centre, $\tilde{\mathbf{x}} := \frac{1}{2}(\tilde{\mathbf{x}} + \mathbf{c})$, until (1.4b) is satisfied. Then include $\tilde{\mathbf{x}}$ into X.

Step 1: Terminate if no better solution was found in five subsequent algorithm steps.

Step 2: Find the worst (in terms of performance value) point in the current set X and call it $\mathbf{x}_h$. Calculate centre $\mathbf{c}$ of $X \backslash \mathbf{x}_h$. Compute a trial point $\tilde{\mathbf{x}} = \mathbf{c} + \alpha(\mathbf{c} - \mathbf{x}_h)$, where α is some reflection ratio. Modify $\tilde{\mathbf{x}}$ by setting values of its elements violating bounds (1.5a) to those bounds values.

Step 3: Check if $\tilde{\mathbf{x}}$ satisfies also implicit constraints (1.4b). If not, move $\tilde{\mathbf{x}}$ gradually towards complex centre, $\tilde{\mathbf{x}} := \frac{1}{2}(\tilde{\mathbf{x}} + \mathbf{c})$, until (1.4b) is satisfied.

Step 4: If $f(\tilde{\mathbf{x}}) < f(\mathbf{x}_h)$ then set $X = \{\tilde{\mathbf{x}}\} \cup X \backslash \mathbf{x}_h$ and go to Step 1. Else set $\tilde{\mathbf{x}} := \frac{1}{2}(\tilde{\mathbf{x}} + \mathbf{c})$ and go to Step 3.

It is clearly seen that the actual complex centre is the ultimate instance and remedy in case of insufficient progress as well as constraints violation — the trial point $\tilde{\mathbf{x}}$ always converges there. Besides, no randomisation takes place after the algorithm has initialised — no wonder then that COMPLEX is less explorative than CRS. COMPLEX is recommended to be run several times in the final stage of optimisation, since results from its single run only may turn out to be unreliable.

Tabu search

Tabu search [48] was proposed rather as a tactics ensuring effective operation of some local search algorithm, by discouraging costly re-evaluation of $f(\cdot)$ at trial points already 'examined'. Tabu search keeps history tables for the solutions found so far, and utilises them to guide the local search procedure so that getting trapped into a local solution may be avoided. In its rudimentary version, tabu search routine is as follows (index k denotes the current algorithm iteration):

STEP 0: Initialise with some initial solution $\mathbf{x}_k$, $k = 0$, with the best solution $\mathbf{x}_l = \mathbf{x}_0$ and with an empty list A of forbidden local algorithm 'moves', i.e. pairs of consecutive solutions $(\mathbf{x}_j, \mathbf{x}_{j+1})$.

STEP 1: Let local routine generate new solution candidates $\{\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_N\}$. Evaluate $f(\cdot)$ for each of them.

STEP 2: Find best solution candidate not being result of any recorded move;
$\tilde{\mathbf{x}}_l = \arg\min_{\tilde{\mathbf{x}}_i,\, i=1,\dots,N} f(\tilde{\mathbf{x}}_i)$, $(\mathbf{x}_k, \tilde{\mathbf{x}}_i) \notin A$. Set $\mathbf{x}_{k+1} = \tilde{\mathbf{x}}_l$ unless there is some unusually good $\tilde{\mathbf{x}}_j$ in A; in such case accept $\mathbf{x}_{k+1} = \tilde{\mathbf{x}}_j$ unconditionally.

STEP 3: Set $\mathbf{x}_l = \mathbf{x}_{k+1}$ if $f(\mathbf{x}_{k+1}) < f(\mathbf{x}_l)$. Insert $(\mathbf{x}_k, \mathbf{x}_{k+1})$ at the beginning of A; truncate A to a predefined length. Go to STEP 1.

The above routine prevents frequenting old optimisation paths save cases when this brings really improved solutions (which might be important in stochastic optimisation). The list A of tabu moves is updated in every iteration so that 'stale' moves expire. This basic scheme is usually supported with some kind of long-term memory in the form of, say, a list B, recording from the very beginning visiting frequencies for each solution approximate $\mathbf{x}_k$. Usually, A and B bias the selection of $\{\tilde{\mathbf{x}}_1, \dots, \tilde{\mathbf{x}}_N\}$ so that non-visited points are preferred.

Tabu search is a very flexible idea; much of its properties depend on the exact strategies for forbidding and promoting trial points, and on the kind of the local search routine controlled by tabu search. Tabu search is used mostly for discrete problems, although there exist guidelines, by its author, for applications to continuous problems. The crucial tabu search topic is the definition of a neighbourhood of some trial point; the choice of space topology may largely determine the routine performance — like in case of evolutionary algorithms, the next and last to be presented in this section.

EA's — Evolutionary algorithms

It is a large family of heuristic routines whose construction was inspired by natural evolution of species observed in the nature. The paradigm of natural selection of better fit individuals is utilised in the context of minima searching, with trial solutions corresponding the individuals and with performance function corresponding individual's fit to environment. Evolutionary algorithms [7, 73, 12] work by iterative modifications of a pool of points (called population) in reproduction, genetic operations and succession phases. Reproduction creates a new pool of trial points (called offspring) from the current pool. Next, the offspring is subject to genetic operations. Finally, in the succession phase, individuals are chosen from the modified offspring and constitute the new generation. Termination criteria may depend on a specific problem, but mostly they are similar to those for direct search algorithms presented above. EA framework can be put as follows:

Step 0: Make the initial population X_k, $k = 0$. Evaluate $f(\cdot)$ for each element of X_k.

Step 1: Stop if some termination criterion met.

Step 2: Reproduction. Produce the offspring O from X_k. Reproduction process is usually biased to promote better fit X_k elements to the offspring.

Step 3: Genetic operations. Modify O by application of various operations. (Variety of operations can be very big: from exactly imitating processes made known by genetics to developing operations tailored to a concrete problem.)

Step 4: Succession. Qualify individuals from O that will form the population X_{k+1}. Evaluate $f(\cdot)$ for each element of X_{k+1}. Go to Step 1.

According to the degree processes observed in nature are imitated, EA can be classified into genetic algorithms, evolutionary strategies, and others (e.g. evolutionary programs, genetic programs). In genetic algorithms the reproduction phase is mostly accomplished by application of roulette-wheel selection mechanism, which diversifies probabilities of an individual being promoted from X_k to O, depending on that individual's fit. Genetic operations used are crossover and mutation; they assume attributes of an individual coded binary to be the chromosome — and act traditionally. Qualification is just setting $X_{k+1} := O$. Evolutionary strategies work similarly, but the coding of an individual's attributes and the genetic operations are rather adapted to a given class of problems.[3] There exist many evolutionary approaches to narrow classes of problems. They are loosely related, in program structure and in applied operations, to genetic algorithms. Those specialised algorithms have been developed e.g. for the problem of a finite automaton design, or for the problem of optimal algorithm design [7, pp. 18–19].

Broad literature on EA's proves their efficiency, robustness and innate stability — they give encouragement for self-made algorithm amendments and experiments. Evolutionary algorithms are global, have support for box constraints on $\mathbf{x}$, and the remedy for a 'child' violating constraints on $\mathbf{y}$ can be given instantly: discard the one and try another. Random nature of most operations is an ally here.

[3]To give an example, in case of continuous optimisation problems, the mutation is disturbing an individual coordinates by a random vector; the crossover is calculating a centre of parents' locations.

3.3 General-purpose simulation-optimisation solvers

With the main goal of solving problems from Chapter 2 on mind, it would be valuable to see how the routines presented above are utilised in professional solvers. The description of selected optimisation tools follows here. The selection was made with purpose to present first much appreciated *AMPL* and *GAMS* modelling and optimisation environments to see how they relate to the specifics of our problems; then to browse through less general but better fitting solvers, and to end up with description of a tool seeming to be the fittest to our case. The most desired solver properties are: co-operation with the user-supplied modelling module, no need for any derivatives to be supplied, support for simulation crashes.

AMPL and *GAMS*: various optimisation routines, one modelling language

AMPL and *GAMS* are competing complete modelling and optimisation environments. They both provide advanced proprietary languages the model and problem definitions must be expressed in. Once this is done, the user may try employing a vast number of offered routines for the optimisation task. Therefore, both *AMPL* and *GAMS* provide a unified way of accessing widely differing solvers, from classic LP to e.g. mixed integer nonlinear programming with discontinuous derivatives [2, 22, p. 98].

Specifically, nonlinear solvers are suggested for the problems of our concern. Not accidentally, the first one proposed by *AMPL* as well as by *GAMS* is *CONOPT* [35] — a commercial GRG implementation. Another solver available in both environments is *MINOS* [22, pp. 201–224] — a projected Lagrangian algorithm working in fashion similar to SQP. There are a number of others, which can be tested effortlessly with no need to re-write the interface between the model and the solver. The decision which one to choose is left completely up to the user; if none is chosen, then a default routine is employed. No automated and adaptive routine selection strategy is implemented.

AMPL and *GAMS* are praised for being excellent tools for rapid prototyping and appropriate solving technique selection. However, they are completely unfit to handle the optimisation jobs described in Chapter 2. First, because of the requirement that the model *must* be coded in a specific language.[4] Second, even if a problem is to be coded in using that proprietary language, none of the underlying solvers has support for exceptions ensuing during the model

[4]Actually, this could be done in the case of simplified IP market model. *AMPL* and *GAMS* languages have loop and branch constructs; they also support tabelarised expressions with tables located conveniently outside the main model-and-problem definition input file.

computation (e.g. the domain errors for square root or logarithm functions). As a matter of fact, for some solvers [35, pp. 15–16] such undefined result gets replaced by *GAMS* with 'some appropriate real number' — but this is considered to be the last resort, and the optimisation terminates if such cases reappear.

Simulation-optimisation tools

GAMS, *AMPL*, and alike, are excellent for prototyping and educational purposes. However, when business and industrial applications with closed-source modelling modules come into play, a niche appears for optimisation tools designed for specific classes of problems. Those tools fall in middle way between *GAMS*-like environments and in-house adaptations of standard routines for specific problems that were cited throughout this chapter and Appendix A. From a number of such tools, *CONSOLE-OPTCAD* [39], *SPRNLP/SOCS* [106], *DOT* [34], *OptdesX* [83] and *OptQuest* [81] have been chosen as good representatives.[5] All of them provide an interface to user-supplied simulation routine, which is supposed to return only the value of the performance index. No explicit gradient information is required from the simulator; all the derivatives necessary for internal optimisation routines are estimated. As regards used optimisation routines, they vary accordingly to class of problems handled by a tool. Most of above mentioned products, except for *OptQuest*, are design optimisation tools. They use all-feasible SQP variants (*CONSOLE-OPTCAD*, *SPRNLP/SOCS*), alternatively accompanied with GRG (*OptdesX*), or with GRG and SLP (*DOT*). *OptQuest* inclines towards solving business, financial and scheduling problems, being rather discrete tasks with indeterminism involved. The optimisation routines used are tabu search and scatter search — another global optimisation algorithm by the tabu search inventor.

It must be emphasised that none of the above tools supports simulation failures. It does not mean, however, that their creators are not aware of reality. *OptdesX* manufacturer writes [83, p. 177]: 'The vast majority of program crashes occur inside the `ANAFUN`[6] subroutine. This happens when either the analysis software has not been fully debugged, or *OptdesX* drives to some design that the analysis routine cannot handle [...].' However the only suggested there remedy to such a situation is to restrict search space even more, and to debug the simulator software. It is important to remind here that constraining the decision variables too eagerly may cut off valuable regions from the domain; it is also worth mentioning that in most cases the simulator routine is in fact a black box, and no code modifications are possible.

[5] A place worth suggesting if one is going to investigate simulation-optimisation tools is the home page of *NEOS* — a powerful network optimisation service [78].

[6] That is the pre-defined name for user-supplied simulation routine.

Despite its limitations and quite old technology, *OptdesX* should be distinguished among the selected tools merely for its creators' awareness of the nature of third-party simulators. As it turns out in the following section, supporting simulation crashes was not the idea to be given up completely.

Epogy: a complete design optimisation solution

Epogy [111] can be rightfully classified as yet another simulation-optimisation tool. However, it possesses a number of valuable advantages that make it deserve a separate section. Needless to say, it has a number of built-in optimisation routines, supports user-delivered simulation code, and, whenever necessary, performs estimation of performance index derivatives. What distinguishes *Epogy* are three things:

- The most appropriate optimisation routine to be run (and the values of that routine parameters) is chosen automatically;
- The optimisation runs in stages, i.e. the solution obtained from one routine is utilised as the start point for another one;
- Unified support for simulation crashes is implemented.

LP, Nelder-Mead simplex search, EA, SQP and Monte Carlo are the built-in *Epogy* algorithms. It is evident that they complement mutually and, run in sequence, can cope with most of design optimisation problems. *Epogy* innovation is the automatic guess being made about the type of $f(\cdot)$ — in case when no hint is given by the user — followed by automated selection of the best fitting routine. Moreover, routine parameters (e.g. termination criteria or differences used in gradient estimation) are also adjusted adaptively. This mechanism is driven by a sort of superior genetic algorithm (mentioned in [111] only vaguely). The mixture of basic algorithms that *Epogy* manufacturer reports to behave particularly well, for the kind of problems like ours, is EA followed by Nelder-Mead search, although a user is free to suggest an arbitrary algorithm sequence.

An important *Epogy* feature is the simulation interface — in particular the assumption that simulation is liable to fail. If *Epogy*, while parsing simulation output files, fails to retrieve a complete $\mathbf{y}$, some default values (supplied earlier by the user) are used to calculate $f(\cdot)$. Such faked performance index value at defective trial points is usually much higher than at neighbouring 'correct' designs [112]. The important lesson that can be learned from *Epogy* is that a successful search for a solution can be performed without any special arrangements in the optimisation routine concerning cases when simulation fails. At least, such is might be the first conclusion — there will be no next one since the detailed description of *Epogy* algorithms is not available.

3.4 Conclusions on others' and the author's approach for problem solving

The following conclusions can be made about existence of simulation-optimisation problems, and about applicability of optimisation approach followed (or suggested) by other authors. First, it becomes quite evident that optimisation problems with performance index determined by simulation output have divided into two branches. The first, widely called just 'simulation optimisation' refers to the class of problems with intrinsic model indeterminism, and with decision variables often taking values from discrete sets. The other branch, frequently termed 'design optimisation', emphasises the simulation noise (the effect of sophisticated simulator numerics) as being the main problem.

Depending on the assumed properties of the problem (differentiability, convexity, monotonicity), various optimisation algorithms are applied in both branches. Since an algorithm working always relies on some assumptions about the problem, and exploits those assumptions in order to reach maximum efficiency, it is important to be realistic while deciding as to what algorithm to apply to a certain problem. Not surprisingly, for problems considered as difficult, direct search algorithms are used equally widely in both branches. In order to mitigate computational effort, rough and fast alternative models are the commonplace [87, 47] that assess the performance index value before (or instead) calling the accurate simulation. In cases when the problem appears less difficult, either naturally or after model simplifications, the applied optimisation routines divide in two branches. For stochastic optimisation, either stochastic approximation is exercised for continuous problems, or direct search is still applied for discrete problems. For design optimisation, gradient routines are applied.

So far, no off-the-shelf optimisation tool has been found that would fit all our problems presented in Section 2, and particularly the problem of power plant set-point optimisation — due to simulation failures. Such conclusion comes as no surprise since it is widely emphasised in the literature that engineering problems require deep initial insight and analysis prior to selection and application of an optimisation routine. However, one of the tools, *Epogy*, exhibits features that place it closest to the direct application to our problems. Those features are also instructive if development of own optimisation approach is considered.

Such is the course taken by the author: to develop his own optimisation approach for the class of difficult problems. There are several reasons to do that. The first is scientific research flavour such development would certainly have. The second, is freedom such research would offer; being confined e.g. to what *Epogy* — the best fitting tool — imposes and keeps secret, could become a limit. The third and last, the 'discovery' of *Epogy* happened in the middle of research, when most of the numerical routines and — last but not least — the laborious simulator interfaces had been implemented. Such author's course is directed by a proposed alternative approach to solve difficult simulation-optimisation problems, which follows in the next section.

Proposed approach to solve simulation-optimisation problems of unknown nature

Having outlined the complications encountered by the author in practical optimisation problems and the current perception of simulation-optimisation by scientific and engineering society, it is the right time to start argue Thesis 2. Ideas of Thesis 2 appear as much intuitive, common-sense and obvious as unprovable by mathematic apparatus. This is because they postulate heuristics (for algorithm selection, modifications and switching) to be placed on the top of other heuristics, which present the selected algorithms themselves. Obviously, the postulated overlying heuristics may not be efficient for some malicious practical problems — but the main goal of the proposed approach is to end up with robust optimisation algorithm, its efficiency being not a major concern.

Thesis 2 is specified below by proposition of an approach to solve simulation-optimisation problems with implicit and feasibility constraints and with the performance index of generally unknown properties. The proposed approach is as follows:

1. Use as much *a priori* knowledge of the problem as is available in order to select a couple of standard optimisation routines out of those well recognised (presented in Chapter 3). It is suggested that selected routines be simple to modify. The routines will be run sequentially, the next one taking over the result from the previous one. Take into consideration availability of the parallel computing environment; in such case prefer rather inherently parallel or easily parallelisable routines. Selected routines should jointly be able to support problem characteristics in all stages of optimisation, which are: initial search for global minimum, intermediate search with support for constraints, final search with linearised both performance index and the constraints.

2. Try to obtain a model and formulate optimisation problem simpler than the original one (e.g. by taking away a part of modelled system) but preserving the original model

features. Use this model for testing, exact switching criteria construction and *ad hoc* algorithm modifications.

3. Prepare appropriate programming interface between simulation and optimisation modules, insulating them mutually in case of a failure of either of them. Apply techniques allowing easy porting to parallel execution.

4. Run chosen routines with rather tight termination criteria to obtain some good solution that will become the reference point in further tests. Observe problems appearing in the run: simulation failures, infinite loops in optimisation, premature termination. Investigate what may be their reasons. Consequently, make modifications to routines as well as to simulation-optimisation interface. Examples of such modifications may be: support for failures, implicit constraint violations. Also, adjust algorithm parameters, mainly the termination criterion. Employ other optimisation routines as new important problem features are revealed that cannot, or should not be, digested by the routines being currently in use.

5. Work out efficient criteria for switching to another routine in the sequence. The hybrid should in general be robust and effective, and the criteria should first take into consideration whether the model can already be supported by the next routine in the sequence. Efficiency is the secondary issue.

6. Make use of parallel environment (if available) and apply the constructed hybrid optimisation algorithm to the original problem or to other problems of the same class.

The above approach is reflected in the account of solving the three practical problems that follows in Chapter 4 and Chapter 5.

Chapter 4

Solution of power plant set-point optimisation problem

Taking as an example the problem of power plant set-point optimisation, one may try to point out most frequent causes that make the optimisation task unnecessarily difficult. Therefore, it is shameful but necessary to admit here that complications in optimisation may sometimes have their origin in

- Errors in simulator operation — e.g. infinite loops that may be entered for certain simulation inputs without any preliminary checking;
- Inability of the simulator to work autonomously, i.e. without human intervention;
- Very little or no *a priori* knowledge of the problem;
- Incorrect optimisation problem formulation — e.g. specification of redundant decision variables or classification as decision a variable unrelated to the performance index value, or incorrect constraints specification.

The reasons for such situation can be twofold. The more common case occurs when the party interested in coupling simulation with optimisation is the modelling module author who has got used so much to simulator imperfections that they do not disturb him in manual simulator operation. Moreover, the simulator author is little acquainted with optimisation theory and practice — and the problems result. The more extreme case is a user equipped with off-the-shelf simulation software, and determined to combine it with optimisation routine quickly and having as little to do with the problem specifics as possible.[1] Unfortunately, such observation

[1]Obviously, in both cases the difficulties would not appear the problem itself were simple. Nowadays, however, this happens rarely — and most problems must be 'tamed' before being submitted to optimisation solver.

itself is not going to improve the situation a bit. Such state is likely to persist — and it appears at various spots in all practical problems considered here. Since IHE simulator is the most affected case it is given here a particular attention.

As it was said, IHE problem is the biggest opportunity — and the biggest challenge — to construct an effective solving approach. It is so because of the simulation failures (confirmed already by manufacturers of *OptdesX* and *Epogy* [83, 111] to be commonplace in optimisation practice) and because of freedom for experimentations with any code running outside the simulator. No other of the considered practical problems offered such working conditions.

Algorithm initial selection

Let it be reminded that at the time initial optimisation algorithm selection was being made no detail about the problem nature was known[2] except for the problem specification itself and for the target operating conditions, which were as follows. The software is planned to be run in a plant control room, and therefore no parallel computing environment is assumed. The software is to help the plant controlling staff in decision making, and not to operate autonomously in a real-time system, so its speed is not a crucial issue. Nevertheless, if the operation speed is to be unsatisfactory, creation of distributed environment, e.g. by purchasing more computer units, is possible.

Like it was presented in Section 2.1 in case of power plant set point optimisation, there is a simplified model at disposition. Test model corresponds to its original, the plant model, with the exception that it is deprived of three one-stage turbines and the accompanying devices like regeneration system or collectors. The pumps have been removed, too. However, most nonlinear elements, i.e. turbines (and, potentially, boilers) remain as well as remains performance index formulation. Model simplification reduces the search space dimension from 21 to 9, and the average simulation time from 0.5 to 0.2 sec. Such simplified model is used here for all optimisation tests. Next, the resulting hybrid algorithm is verified for the target full size model.

The initial choice of algorithms is affected by pre-existent competitive and complete solutions in the field, and by specifics of IHE simulator. Most of the commercial competitive products[3] for power systems modelling and optimisation use system elements described by far

[2]The characteristics of performance index function given in Section 2.1 are only the effect of optimisation attempts. They should be concerned as a kind of optimisation results; however, they have been presented first to put more emphasis on the phenomenon of feasibility constraints, and on their significance.

[3] An example of a complete business solution for advanced process modelling and control is *NOVA*, by PAS, Inc. [84]. It represents the line of modelling and optimisation tools that was introduced in Section 3.3; the

simpler models than those of IHE. Competitive products performance is satisfactory only for weakly coupled turbosets, e.g. those not interlinked by common steam collectors, unlike ours. Simple system structure adds to element model simplicities and sometimes allows for application of linear optimisation routines, which is also the approach suggested earlier by IHE itself [95, pp. 95–114]. Instead, IHE modeller applies nonlinear models — and nonlinearity may breed local optima. Considered that, one global non-gradient routine is chosen for preliminary optimisation, and one local gradient routine is chosen for final optimisation stage. The task of the former one is to overcome consequences of nonlinear model application; the latter is to operate on a linearised model hopefully expected to behave like its commercial counterparts.

CRS2 (cf. p. 73) is chosen as the global routine. It requires minimum knowledge of the problem, no start point and, after the initialisation phase, generates new solutions one by one (i.e. not in flocks like EA does). Simultaneously, its parallel implementation, if need be, is straightforward. Other direct search routines are discarded, either for not being global (Powell) or 'not global enough' (COMPLEX), or for their wide-ranging applicability covering discrete problems (tabu search, simulated annealing) — possibly at the cost of degraded performance.

The choice of local gradient routine is determined by the problem formulation. The ratio of the numbers of decision and dependent variables, and the way constraints are specified, makes GRG the unbeaten candidate. SLP is excluded because of infeasible solution approximations it produces, which could jeopardise its working given high model nonlinearity. SQP exclusion happens for the same reason, especially that it makes quadratic objective approximations, which would probably be all misleading in our case. It is hoped that in the vicinity of the optimum gradients of performance index and of constraints could be easily estimated by e.g. finite difference formula, and such estimation will hold in sufficiently big region.

model is defined in a proprietary language (with an option to plug in user-supplied routines), the optimisation is driven by a sort of SQP solver adapted to handle non-smooth problems that are admitted by the manufacturer to appear ubiquitously in industry and business. The investigation made by IHE shows that PAS products adapted and offered by Honeywell for plant control use much simplified models as compared to those considered here. *NOVA* language has been also used by IHE for direct implementation of IHE modelling methodology. Since model solving in *NOVA* utilises (2.3), failures in (1.3) solving do not leave any trace as to which model variable could have caused the simulation failure. Moreover, *NOVA* optimisation engine has turned out to be very vulnerable to such failures, and very sensitive to the location of the start point. Generally, it is felt by IHE that most important parts of modelling and optimisation are carried out in *NOVA* without possibility of being inspected by a user.

Termination criteria selection

A cursory look at presented optimisation routines specifications brings a conclusion that they are often more precise about some internal parameters, usually set by the rule of the thumb, than about the termination criterion itself. In this regard much is left to the user, although there is a trend (in case of design optimisation tools — cf. p. 79 and following) to relate it to user-supplied estimate of $f(\mathbf{x}^\star)$. There is also another reasonable habit to let the algorithm work until no progress is made in some time window. Apart from that, there are the usual absolute-improvement or relative-improvement criteria comparing $f(\mathbf{x}_k)$ and $f(\mathbf{x}_{k+1})$.

Since for set-point optimisation problem, similarly to optimal pricing problem, no $f(\mathbf{x}^\star)$ estimate is delivered, the relative-improvement termination idea is dropped. Instead, the ideas of time-window improvement and absolute improvement are adopted and consequently applied for all tested routines. Therefore, an algorithm stops if:

- Performance index evaluated at the best point in the pool differs from performance index evaluated at the worst point in the pool by ϵ_{A} (absolute accuracy);[4]

- Performance index evaluated at the current solution estimate differs from performance index evaluated at the solution estimate w iterations ago by less than ϵ_{I} (minimum improvement);

- The number of performance index evaluations exceeded k_{S}.

4.1 Optimisation tests and algorithms modifications

CRS adaptations to simulation failures; results

First of all, CRS[5] was started with no initial point given and with virtually no termination criterion, save $k_{\mathrm{S}} = 100,000$. Such tests of very high computational budget were done to find a point that might be the test problem solution. Unfortunately, at the very beginning of the optimisation process sudden simulation failures broke the computation altogether. Modifications had to be made then as well in the simulation-optimisation interface (to intercept failure signals) as in CRS itself, to support such cases. There are two possibilities of handling such failure: either to drop the trial point for which the simulation fails (i.e. treat it the same way

[4]Criterion used only if applicable.

[5]For CRS, implicit constraints are represented by penalty function calculated using city metric norm of $\mathbf{r}(\mathbf{a}_{\mathbf{y}}, \mathbf{y}, \mathbf{y}_{\mathrm{L}}, \mathbf{y}_{\mathrm{U}})$ with $\mathbf{r}(\cdot)$ defined as in (2.7), which adds to the actual value of $f(\cdot)$.

$\epsilon_{\mathrm{A}} \backslash \bar{\bar{X}}$	$32(\dim \mathbf{x}+1)$	$16(\dim \mathbf{x}+1)$	$8(\dim \mathbf{x}+1)$	$4(\dim \mathbf{x}+1)$
10^{-6}	**1.648**	**1.647**	**1.764**	**1.893**
	100000	*100000*	*100000*	*33760*
10^{-5}	**1.648**	**1.648**	**1.720**	**1.875**
	87840	*76810*	*72071*	*4652*
10^{-4}	**1.648**	**1.648**	**1.733**	**1.924**
	92011	*53949*	*36893*	*3940*
10^{-3}	**1.666**	**1.691**	**1.850**	**1.897**
	58478	*32453*	*2969*	*820*
10^{-2}	**1.812**	**1.828**	**1.855**	**1.922**
	4141	*1977*	*927*	*404*
10^{-1}	**1.853**	**1.872**	**1.876**	**1.942**
	1898	*894*	*442*	*235*

Table 4.1: Results of test problem optimisation with CRS algorithm for various number $\bar{\bar{X}}$ of points in the pool and for various value of ϵ_{A} in termination criterion. The average value of $f(\cdot)$ (bold) for CRS solutions and the average number of $f(\cdot)$ evaluations (italics) are calculated for each ϵ_{A} and $\bar{\bar{X}}$ combination from four CRS runs. (The average of *100000* denotes that the maximum number k_{S} of $f(\cdot)$ evaluations has been exceeded at least once in a series.)

as points violating explicit constraints), or to accept the point assigning it infinitely high value of $f(\cdot)$. Mode of handling simulation failures is made an extra CRS option.

With those changes, CRS was run for each combination of several ϵ_{A} parameter values and different number of points in CRS pool. The average $f(\cdot)$ value and average number of $f(\cdot)$ evaluations for this experiment are given in Table 4.1. Those results should be considered as very rough estimates of CRS performance — first, because of the small number of optimisation repetitions, and second, because CRS is not fit nor designed to carry out the optimisation process to the very end, especially if the solution is to lie on the constraints. Nevertheless, moving diagonally across Table 4.1 one can easily see that the solution quality is obtained at the cost of many evaluations of $f(\cdot)$. There is only the question where lies the reasonable compromise between them.

ϵ_A \ on failure	reject $\tilde{\mathbf{x}}$	assume $f(\tilde{\mathbf{x}}) = \infty$
10^{-6}	1.750	1.739
10^{-5}	1.751	1.733

Table 4.2: Closer investigation of Table 4.1 results: optimisation results for pool size $\bar{\bar{X}} = 8(\dim \mathbf{x} + 1)$, various ϵ_A's and rejecting faulty trial points, or setting $f(\tilde{\mathbf{x}}) = \infty$. Average values of $f(\cdot)$ for CRS solutions are calculated in each case from 20 optimisation repetitions.

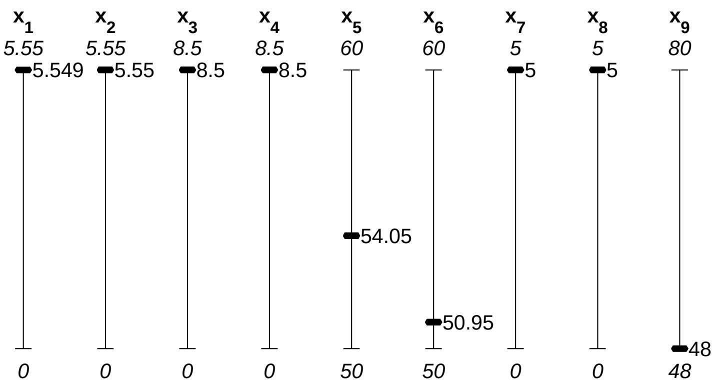
ϵ_A \ on failure	reject $\tilde{\mathbf{x}}$	assume $f(\tilde{\mathbf{x}}) = \infty$
10^{-6}	1.892	1.786
10^{-5}	1.895	1.889

Table 4.3: Closer investigation of Table 4.1, results made as for Table 4.2 for CRS with pool size $\bar{\bar{X}} = 4(\dim \mathbf{x} + 1)$.

Since increasing pool size over $16(\dim \mathbf{x} + 1)$ apparently does not improve the solution quality dramatically, impact of the way simulation failures are handled was tested more profoundly for moderate CRS pool sizes. Table 4.2 presents average performance index value for pool size of $8(\dim \mathbf{x} + 1)$ points, and Table 4.3 — for a pool of $4(\dim \mathbf{x} + 1)$ points. Generally, accepting unlucky points as infinitely bad seems to perform slightly better than rejecting them altogether. By the way, from closer examination of ϵ_A impact it comes that setting it to 10^{-6} or to 10^{-5} virtually does not influence solution quality.

Let us discuss the best solution obtained so far. It is (cf. Fig. 2.3)

$$\mathbf{x} = \begin{bmatrix} 5.549 \\ 5.55 \\ 8.5 \\ 8.5 \\ 54.05 \\ 50.95 \\ 5.0 \\ 5.0 \\ 48.0 \end{bmatrix} \tag{4.1}$$



and the value of $f(\cdot)$ there is 1.646. For the convenience of interpretation, the values of decision variables have been put in (4.1) on rulers. First, it is important to note that CRS is, in general, a robust routine for it — started without initial point — is able to find a feasible solution. Average performance index value at that solution is 2.1. Provided big budget, CRS is in position to find solution much better than that. For solution (4.1) flows through reduction valves (not shown since they are dependant variables) are virtually zero — this means that all steam, before it goes to the industrial receptions, is utilised in turbines for electricity generation. Simultaneously, cost of steam generation is kept as low as possible by x_9, powering less efficient boiler, set to lower bound. Similarly for turbines, the more efficient one receives more steam; $x_5 > x_6$. A note must be made on x_5 vs. x_6 that no further reduction of x_6 can be done in favour of x_5 due to implicit constraint violation. This constraint is illustrated by a graph of feasible area in Fig. 4.1. Worse solutions found in other optimisation runs make the steam flow through reduction valves or do not load more efficient boiler and turbine appropriately.[6]

The solution (4.1) is also better than the test problem solution found by *Epogy* package.[7] Location of the latter one is basically the same, except for flows x_5 and x_6, equal 53.25 and 51.74, respectively. The best performance value found by *Epogy* is 1.650. Such value has been found after some 25,100 evaluations of $f(\cdot)$, and no further improvement was detected in the following 20,000 evaluations. Like for (4.1), *Epogy* solution is also located on the brink

[6]The test problem and its solution have been presented in limited scope in [61].

[7]The trial version of *Epogy* package that was used in the optimisation experiment considerably limits the number of simulation input and output variables. That is why it could be tested only for the test IHE problem. The product was provided with an infeasible initial solution; simulation failures were represented by a very high value of the performance index there; no suggestion about the solving methodology or the problem nature whatsoever was specified on *Epogy* solver startup.

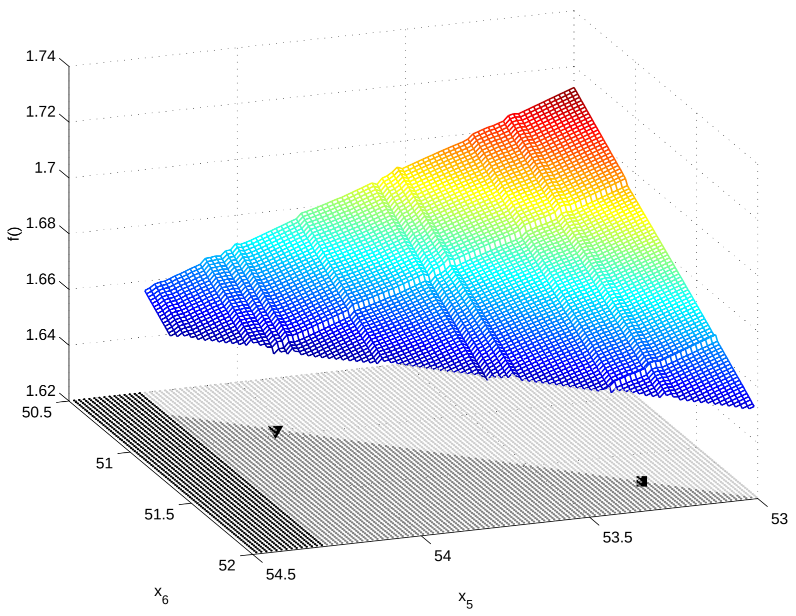


Figure 4.1: Performance index surface plot for the test problem and for x_5 and x_6 (above). Plot of regions where various constraints get active (below; the meaning of colours is the same as in Fig. 2.4). CRS solution (4.1) location is marked with a triangle, and *Epogy* solution is marked with a square.

of implicit constraints area (cf. Fig. 4.1), nevertheless *Epogy* professional optimisation solver was apparently not able to 'glide' along it in pursue of a better point. Such outcome of the optimisation experiment strengthens the general notion that solving simulation-optimisation problems is a tough task, in spite of a number of worked out pre- and post-optimisation techniques (e.g. problem reformulation, response surface methodology, design of experiments, solution validation etc.)

EA involvement and adaptations to the problem; results

Inefficiency of CRS application for solving test problem as a whole tempts to use EA for the same purpose. It is particularly interesting to check if applying EA with parameters generally considered to make it robust, will bring any qualitative improvement. Especially, it is of interest whether EA is able to find a solution as good as found by CRS, but with considerably smaller computation budget.

The evolutionary algorithm that is chosen for our problem is in fact an evolutionary strategy. The details of its implementation and selection of parameters values were advised by colleagues, deeply and for long time involved in practical applications of EA's for both power system [33] and for microwave circuit [8] optimisation problems. The emphasis in algorithm design was put on its simplicity and generality, and not on its adaptation to any particular problem. Therefore, trial point coordinates are assumed the 'genome' of an individual. Consequently, appropriate genetic operations, crossover and mutation, are adopted: the crossover produces a 'child' which is a centroid of its 'parents', and the mutation is disturbation of 'child' coordinates by some random variable. The reproduction is accomplished trough a series of comparisons performed in pairs which were randomly chosen from the original population. The succession accepts unconditionally the offspring; the termination criteria are the same as for CRS (see p. 87). In detail, the relevant steps of the original scheme (cf. p. 73) are as follows:

STEP 2: Make $O = \{O_\mathrm{t}, X_k\}$, where every element of O_t is winner of a tournament.

Tournament. Choose at random, with uniform distribution, two individuals $\mathbf{x}_a$ and $\mathbf{x}_b$ from X_k. The tournament winner is $\mathbf{x}_a$ if $f(\mathbf{x}_a) < f(\mathbf{x}_b)$, otherwise the winner is $\mathbf{x}_b$.

Therefore, O consists of two distinct subsets, the tournament winners O_t and the whole current population X_k.

STEP 3: Set $O := \{O_\mathrm{t}, O_\mathrm{c}\}$, where every element of O_c is created by crossover.

Crossover. Perform tournament N times in X_k to appoint N crossover participants $\mathbf{x}_1, \ldots, \mathbf{x}_N$. The crossover result is a point $\tilde{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_i$.

Modify O by mutating each individual.

Mutation. Set $\mathbf{x} := \mathbf{x} + \boldsymbol{\xi}$ where $\boldsymbol{\xi}$ is a realisation of multidimensional Cauchy distribution, i.e. p.d.f. for ξ_i is $\frac{1}{\pi}\frac{\sigma_i}{\sigma_i^2 + (x - x_i)^2}$.

Set coordinates of mutated points that violate cubical bounds (1.4a) to those bounds.

Settings for our EA are as follows. Population size $\bar{\bar{X}}_k$ is constant through the algorithm run and equal 50, which has been suggested as reasonable compromise between explorability (larger $\bar{\bar{X}}$) and efficiency (smaller $\bar{\bar{X}}$) for the given problem dimension. The number $\bar{\bar{O}}_\mathrm{t}$ of individuals entering the offspring directly is set to 30% of $\bar{\bar{X}}$. The tournament is simplified to the extreme in order to avoid any problem-specific traps that could happen for roulette-wheel algorithm. The number of crossover participants has been suggested $10 \div 20$, and it was decided to implement the algorithm with the lesser value to improve explorability. Individuals bred from crossover make the rest of the offspring, i.e. $\bar{\bar{O}}_\mathrm{t} = 0.7\bar{\bar{X}}$. All the above parameters have been deliberately hard coded into the routine in order to limit the number of parameters required at the start. The adjustable algorithm parameters are all the termination criteria

$\epsilon_A \backslash p$	.95	.995	.9995
10^{-6}	**1.713**	**1.660**	**1.690**
	27225	*72288*	*94413*
10^{-5}	**1.712**	**1.660**	**1.694**
	32350	*52763*	*99075*
10^{-4}	**1.708**	**1.661**	**1.693**
	44763	*70088*	*96613*
10^{-3}	**1.711**	**1.661**	**1.694**
	34388	*61325*	*99050*
10^{-2}	**1.725**	**1.662**	**1.765**
	59675	*69188*	*55400*
10^{-1}	**1.705**	**1.661**	**1.997**
	34275	*44975*	*5638*

Table 4.4: Results of test problem optimisation by EA for various variances of random variable $\boldsymbol{\xi}$ used to mutate the offspring and various value of ϵ_A for termination criterion. The average value of $f(\cdot)$ (bold) for EA solutions and the average number of $f(\cdot)$ evaluations when the last improved solution was found (italics) are calculated for each ϵ_A and p combination from four EA runs.

and $\boldsymbol{\sigma}$ in Cauchy distribution, which is to determine EA convergence speed. For convenience, instead of working with $\boldsymbol{\sigma}$, variance of $\boldsymbol{\xi}$ will be represented here by probability p of the event that $\frac{x_{L,i}+x_{U,i}}{2} + \xi_i \in < x_{U,i}, x_{L,i} >$ for a single i, i.e. that i-th coordinate of a mutated centrally located point will stay within the bounds. Therefore, σ_i is determined by p with the formula $\sigma_i = \frac{x_U - x_L}{2\tan(p\pi/2)}$, for all i.

Occasional simulation failures have enforced EA modifications similar to that implemented in CRS. Therefore, the failure at $\tilde{\mathbf{x}}$ can result in $\tilde{\mathbf{x}}$ rejection (in such case the operation that made $\tilde{\mathbf{x}}$ is repeated) or in setting $f(\tilde{\mathbf{x}}) = \infty$.

Results of EA invocation for the test problem and with parameters as for CRS, where applicable, are presented in Table 4.4. The major change in configuration of tests underlying Table 4.4 data, when compared to Table 4.1, is that EA was tested for varying mutation variance rather than pool size. Moreover, the numbers in italics denote last $f(\cdot)$ evaluation when progress was made. Such progress reporting method results from the fact that EA commonly exceeds $k_{\mathrm{S}} = 100,000$. This should come as no surprise considered that every individual is subject to mutation — this, combined with problem solution lying on constraints makes it practically impossible to satisfy ϵ_{A} criterion, since penalty function may easily get activated. Possibly some other uniform termination criterion should have been chosen in order to compare CRS and EA effectively. However, by investigating the numbers in italics in Table 4.4 one still is able to make observations on the progress EA made. First, small mutations (0.9995) make EA find new, better solutions, almost intermittently (observe small number of italics in the relevant column). Next, despite that $p = 0.9995$ appears to improve solutions steadily, setting $p = 0.995$ yields results better in terms of their average values, although new better solutions are not found so often. Apparently, $p = 0.995$ seems to be best fitting, at least for the test problem. The termination criteria should probably have been determined by ϵ_{I} and w, as they ignore worst pool points and pool dispersion (which is quite natural for problems with redundant decision variables defined).

EA does not prove here superior to CRS, at least as the algorithm to solve alone the problem. They both are costly; moreover, EA does not find a solution as good as CRS does (although $f(\cdot)$ for the best one here was 1.656, and it has been found also for $p = .995$). The major reason for it is, by all means, notorious mutation of EA individuals; this definitely qualifies EA rather for preliminary optimisation phase only.

Gradient estimation results

A series of experiments concerning gradient estimation was carried out with twofold purpose: to find out more about the shape that implicit and feasibility constraints have, and to find out whether they really allow application of some local gradient routine in optimisation final stage. The gradient estimation procedure that was used implemented finite forward difference scheme (A.4), and was enriched with automatic step size adaptation. For each dimension the following steps are executed to estimate $\nabla_{x_i} f(\mathbf{x})$:

Step 0: Initialise with the actual initial step size $c_i = \frac{3}{2}c_i^+$ (c_i^+ is the step size built up by previous estimations), and with the minimum and maximum step sizes, $c_{\min}$ and $c_{\max}$. Set c_i to c_i or $c_{\max}$, whichever less.

Step 1: Compute $\hat{g}_{c_i}(\mathbf{x}) = \frac{f(\mathbf{x}+c_i\mathbf{e}_i)-f(\mathbf{x})}{c_i}$ and $\hat{g}_{2c_i}(\mathbf{x}) = \frac{f(\mathbf{x}+2c_i\mathbf{e}_i)-f(\mathbf{x})}{2c_i}$, the two alternative estimates of $\nabla_{x_i} f(\mathbf{x})$. If $|1 - \frac{2\hat{g}_{c_i}}{\hat{g}_{c_i}+\hat{g}_{2c_i}}| < \epsilon$ then return the estimate and update c_i^+ calculating moving average from last 20 gradient estimations (this one included).

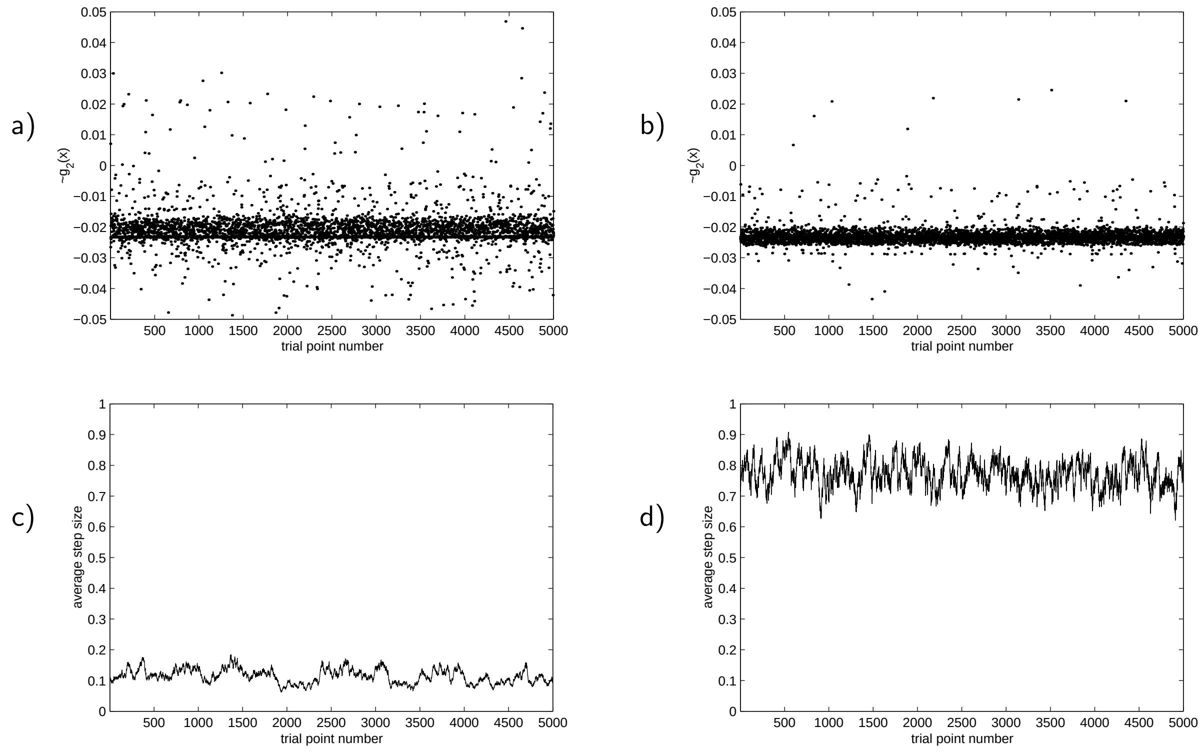


Figure 4.2: Results of performance index gradient estimation for the test problem case. Plots a) and b) present $\nabla_{x_2} f(\mathbf{x})$ estimates, and plots c) and d) present the average step size c_2^+ used for gradient estimation. Plots a) and c) present results for small (0.1) initial step size, and plots b) and d) present results for large (0.9) initial step size. Other settings: gradient estimation accuracy $\epsilon = 0.05$, $c_{\min} = 0.01$, $c_{\max} = 1$, history window in which c_2^+ was calculated was set to 20. Results are presented for 5,000 subsequent gradient estimation attempts performed at points randomly selected from the neighbourhood of $\mathbf{x}^\star$.

Step 2: If $c_i < c_{\min}$, return the estimate as calculated in Step 1 and report estimation inaccuracy. Otherwise set $c_i := \frac{1}{2}c_i$ and go to Step 1.

The shapes of feasibility constraints have been already presented in Section 2.1. As regards the application of a gradient routine, optimum proximity (defined by $< x_1^\star - 1, x_1^\star + 1 > \times \cdots \times < x_9^\star - 1, x_9^\star + 1 >$) was sampled with 38,000 gradient estimations. No simulation failures were detected, but the gradient estimation results were unsatisfactory. Fig. 4.2 presents values of gradient estimate in the direction of x_2 and the corresponding graph of the average step size c_2^+, for two initial values of c_2^+ the estimation was started with. The comment can be made on c_2^+ that it always oscillates around its initial value. It means that there is no step size (at least in the considered range) that guarantees an estimate better than others. On the contrary, from what Fig. 4.2a and Fig. 4.2b present, bigger step size guarantees less varying estimates. This can mean that the gradient estimation procedure applied was inadequate to

		minimum and maximum step sizes for gradient estimation			
		$c_{\min} = 10^{-3}$		$c_{\min} = 10^{-5}$	
accuracy of line search	accuracy of $\hat{\mathbf{g}}(\cdot)$	$c_{\max} = 10^{-2}$	$c_{\max} = 1$	$c_{\max} = 10^{-2}$	$c_{\max} = 1$
10^{-2}	10^{-1}	**2.392** *4*	**1.649** *4*	**4.104** *4*	**2.091** *5*
10^{-2}	10^{-3}	**3.221** *3*	**4.176** *3*	**1.757** *3*	**5.652** *4*
10^{-4}	10^{-1}	**1.647** *32*	**1.647** *2*	**1.647** *2*	**1.647** *90*
10^{-4}	10^{-3}	**1.647** *125*	**1.647** *28*	**1.647** *6*	**1.647** *3*

Table 4.5: Test problem solving using a steepest descent gradient search routine with limited line search step size. Table entries denote performance index value (bold) and number of algorithm steps (small italics) found by routine run for various parameter values: step size limits and accuracies for gradient estimation routine, and the overall accuracy for line search (in terms of absolute $f(\cdot)$ improvement). The value of $f(\cdot)$ at the start point was 1.647.

this problem. However, all the above observations do not have much common with the most fundamental obstacle: please note that the gradient *sign* changes occasionally — and this is the real hindrance for applying a gradient routine.[8] Those positive gradient estimates denote, in fact, failed gradient estimations, i.e. those for which no c_2 could guarantee the desired estimation accuracy.

Despite the above discouraging results a simple gradient routine was tested in the solution neighbourhood. It was just a steepest descent scheme with limited step size and golden section line search procedure.[9] It was started several times, with varying parameters, producing results that are collected in Table 4.5. It failed to improve the initial result in two ways — either terminating prematurely (cf. table entries with the number of steps less than 10), or oscillating around the starting point, making no improvement. Actually, in several cases the

[8]There are reasons to suspect that those few gradient estimates reverting the search direction are, in fact, results of simulation noise, and not the result of the real performance index shape.

[9]The golden line search procedure had to be also made immune to simulation failures. This feature turned out to be never used, though, and it will not be presented here.

optimisation result was worse than the initial solution, due to the forced minimal step size in line search routine. Generally, no rule was found that could determine behaviour of the tested algorithm. It cannot be excluded that further experimentations, especially increasing the step size, could help the presented situation. However, it would mean making the local procedure not so local, which would oppose the initial motivation. Possibly the best solution would be to resort to response surface methodology (cf. Section A.2) — but this is not always working (cf. performance index graphs in Section 5.1 and imagine results of e.g. polynomial approximation in that case), and finally was not tested. Negative results of gradient estimation and golden line searches made the use of GRG out of consideration.

COMPLEX involvement and adaptation to the problem; results

It is evident that none of the presented algorithms in its original shape is in position to solve the problem alone. Moreover, the nature of the performance index effectively prevents application of a gradient optimisation routine because no gradient can be safely estimated. Therefore, COMPLEX routine (cf. p. 74) was employed. COMPLEX is not so explorative as CRS or EA, maintains considerably smaller pool of points, and has built-in support for explicit and implicit box constraints — those features fit the routine perfectly to our problem where gradients are not computable.

Before COMPLEX has been applied to the test problem, it was equipped with the same termination criteria as CRS (see p. 87). Similarly, simulation failure during augmentation operation can result in assuming infinitely bad quality of the trial point, or in trial point rejection. (As regards COMPLEX Step 2, where the reflection is made, a simulation failure is considered equal to implicit constraints violation.) It had been only after COMPLEX routine was run with the test problem, that the discovery of nonconvex domain was made. In details, COMPLEX reported simulation failure when $\tilde{\mathbf{x}}$ was approaching complex centre (please note that performance index is never evaluated for complex centre by the original COMPLEX routine). Since $f(\cdot)$ was computable at all other pool points, it had to mean domain non-convexity. A series of sections through the domain have been made as presented in Section 2.1. They confirm that the domain is slashed and spotted with various regions where the IHE simulator simply fails.

Since the guidelines of research were to solve the problem without modifications made to the simulator, workarounds had to be made in COMPLEX routine with purpose to make it robust. Infeasibility of complex centre can be overcome by extra augmentations, as it was proposed in [41]. However, in practice this has turned out to be insufficient: gradual approaching an infeasible complex centre does not guarantee that any feasible point will be encountered on the way. Tests have shown that extra augmentations do not make the augmented complex centre feasible, and the the routine gets stuck exactly where the centre infeasibility happens.

To overcome this difficulty, we propose another improvement, inspired by the way original COMPLEX works, i.e. by shifting a trial point $\tilde{\mathbf{x}}$ to the complex centre. The innovation is that it will be the reflection centre $\tilde{\mathbf{c}}$ (initially, equal the complex centre $\mathbf{c}$) that will be shifted in the direction of the best point in the complex. Such shift will create a new $\tilde{\mathbf{c}} := \frac{1}{2}(\tilde{\mathbf{c}} + \mathbf{x}_l)$, where $\mathbf{x}_l$ is the best complex vertice. After the modifications, COMPLEX routine incorporates both extra augmentation and centre shift operations. Each time a centre $\mathbf{c}$ is calculated by the original routine, the feasibility check for $\mathbf{c}$ is made. Should the test fail in augmentation operation, $\mathbf{c}$ is called $\tilde{\mathbf{c}}$, and $\tilde{\mathbf{c}}$ is gradually shifted until feasibility is regained. Such shifting is executed also if the backtracking after reflection does not bring any solution better than $\mathbf{x}_h$. In all the above cases $\tilde{\mathbf{c}}$ finally gets so close to the best point that its feasibility and $\tilde{\mathbf{x}}$ quality are guaranteed. The only drawback of this workaround can be premature algorithm convergence. However, it is the price one has to pay for robustness. After all, one may start COMPLEX over several times in hope for good luck in complex creation phase.

Table 4.6 presents a comparison of ordinary and improved COMPLEX routines performance. Both routines were started thirty times for each combination of ϵ_{I} and w values, from the first feasible solution obtained in preceding experiments by CRS (i.e. quite far from the optimum).[10] The conclusion can be made that the improved routine version is, on average, more effective: with the same termination criteria, it finds solutions up to 10% better than its classic counterpart. The costs of such efficacy improvement are rather high (cf. the number of performance index evaluations). However, they stay an order of magnitude less than in the case of CRS (Table 4.1) and EA (Table 4.4) when solutions of similar quality are considered.

To verify the modified COMPLEX algorithm ability to carry out optimisation as well as to check the influence of other COMPLEX settings, the routine was run a number of times from the same starting point $\mathbf{x}_0$ as in previous COMPLEX tests. The performance index at $\mathbf{x}_0$ is 2.216. COMPLEX algorithm behaviour against change of two selected parameters is presented in Table 4.7. As far as mean solution quality is considered, the best is to set small pool size, $\bar{\bar{X}} = 2(\dim \mathbf{x} + 1)$. Larger pools tend to have adverse impact on solution quality, but it is not strong. Such result cannot be a coincidence as it appears for almost all ϵ_A considered. One of possible explanations could be that small pool gives the starting point more importance, and since the starting point is already the result of some optimisation, it prevails over those remaining pool points that are selected at random. This reasoning, however, is not based on

[10]The only amendment made to the original COMPLEX routine was that a number of retries was performed in complex creation phase if complex centre infeasibility was detected.

$\epsilon_{\mathrm{I}} \backslash w$	100	50	20
	1.662	**1.658**	**1.708**
10^{-5}	1.739	1.738	1.807
	3973	**3300**	**923**
	1768	1493	1002
	1.668	**1.675**	**1.737**
10^{-3}	1.736	1.750	1.832
	2339	**1354**	**647**
	1240	1269	650
	1.759	**1.786**	**1.897**
10^{-1}	1.862	2.037	2.111
	483	**319**	**190**
	608	286	180

Table 4.6: Table entries present the average performance index values (large font) and number of $f(\cdot)$ evaluations (small font) for test problem optimisation by COMPLEX routines. Bold numbers refer to COMPLEX version with improvements by the author; the standard numbers refer to original COMPLEX version. Test were performed for varying accuracy ϵ_{I} and history window w used in termination criterion (cf. p. 87), and for $f(\mathbf{x}_0) = 2.216$.

$\epsilon_A \backslash \bar{\bar{X}}$	$16(\dim \mathbf{x}+1)$	$8(\dim \mathbf{x}+1)$	$4(\dim \mathbf{x}+1)$	$2(\dim \mathbf{x}+1)$
	1.855	**1.829**	**1.813**	**1.810**
10^{-6}	1.683	1.663	1.646	1.646
	4721	*3158*	*3484*	*2740*
	1.879	**1.874**	**1.871**	**1.845**
10^{-5}	1.658	1.694	1.665	1.647
	4404	*3130*	*3015*	*2059*
	1.868	**1.826**	**1.842**	**1.816**
10^{-4}	1.677	1.666	1.647	1.646
	3419	*3553*	*3405*	*2130*
	1.853	**1.862**	**1.855**	**1.809**
10^{-3}	1.689	1.678	1.673	1.647
	5321	*2789*	*1911*	*2306*
	1.866	**1.917**	**1.823**	**1.858**
10^{-2}	1.670	1.668	1.684	1.647
	2623	*2117*	*1533*	*1518*
	1.951	**1.947**	**1.994**	**1.966**
10^{-1}	1.703	1.671	1.666	1.651
	1905	*1885*	*1569*	*1787*

Table 4.7: Results of test problem optimisation with COMPLEX algorithm run from a point far from $\mathbf{x}^\star$, for varying complex size $\bar{\bar{X}}$ and varying value of ϵ_A. The average value of $f(\cdot)$ (bold) for COMPLEX solutions, the best value (normal) of $f(\cdot)$ and the average number of $f(\cdot)$ evaluations (italics) are given for each combination of ϵ_A and $\bar{\bar{X}}$.

strong foundations. As regards the termination criterion, selecting $\epsilon_A = 10^{-6}$ or $\epsilon_A = 10^{-3}$ does not have any apparent influence on the quality of result. It turns out that this termination criterion serves only the detection that COMPLEX, in the course of gradual contractions, has collapsed completely, and no further progress can be made altogether. Like in case of EA, selection of this termination criteria does not appear to be a fortunate one. Possibly further improvements of COMPLEX could be needed, were it not for the best results obtained in a series of optimisation runs (20 repeated optimisations) carried out for each Table 4.7 cell. They turn out to be virtually equally good for wide choice of ϵ_A and $\bar{\bar{X}}$. Such results confirm the general reputation of COMPLEX as an algorithm requiring to be given multiple 'chances' (i.e. restarts).

The number of simulation invocations needed is, for most Table 4.7 entries, dependent on complex size, as if extra complex vertices were only a needless burden that has to be dragged and that utilises computing resources. Therefore, the original suggestion to stay with small complex size [20], and to devote extra computation budget for COMPLEX restarts, is in force. Nevertheless, the most valuable and general conclusion is that COMPLEX is in position to approach the optimum starting from as far as just some remote point where (1.4) is satisfied.

Hybrid algorithms construction and testing

From the results presented so far, one may come to the conclusion that none of the considered routines can perform the optimisation task alone as long as the computation budget is limited. The necessity of switching from global to a local routine is evident — and cited in numerous optimisation problem descriptions. The 'only' question remains, *when* such switching should take place. The main criterion of switching from global to local optimisation routine that is proposed for the test case is the maximum number of performance index evaluations done by the preliminary routine. As regards the termination criterion for the latter routine, which is COMPLEX, $\epsilon_{\mathrm{I}} = 10^{-5}$ has been selected with the history window $w = 50$. Both settings result from Table 4.6 data analysis; such setting for ϵ_{I} still brings improvement in quality of found solutions and is hopefully still larger than simulation noise amplitude (cf. Fig. 2.7). Setting w to the value of 50 has been considered a good compromise between improved quality (cf. results for $w = 20$) and degraded efficiency ($w = 100$). Including ϵ_{A} as additional termination criterion has been discarded as being irrelevant (cf. Table 4.7). The maximum number k_{S} of performance index evaluations has been set to 10,000, the limit within which good optimisation results can easily be obtained (cf. Table 4.6 and Table 4.7).

Selection of such switching scheme favouring rather early switching (there is no comprehension for lengthy global explorations) is possible only after many tests have been done that gave the feeling for the nature of $f(\cdot)$. Particularly, $f(\cdot)$ is suspected to be unimodal, and the only real obstacle are regions where the simulation fails, i.e. the active feasibility constraints.

Two hybrid algorithms have been created and tested with COMPLEX run in their final optimisation stage; CRS is the preliminary optimisation routine in one of them, EA is the preliminary optimisation routine in the other. Both CRS and EA operate under uniform termination criterion: w is set to 200 and ϵ_{I} to 10^{-5}, but both the settings, especially ϵ_{I}, are rather arbitrary ones, aiming to detect the situation that no progress is made — the point is only how long (w) one can wait for any progress to appear. CRS operates with pool of $8 \dim \mathbf{x} = 72$ points, and it assumes $f(\tilde{\mathbf{x}}) = \infty$ for those $\tilde{\mathbf{x}}$'s where the simulation fails. Such settings are a kind of trade-off between the results from Table 4.2, current knowledge of the problem and suggestions in [91] for the pool size to be at least $10 \dim \mathbf{x}$. EA operates with probability p that a centrally located $\mathbf{x}$ will, after mutation, still be within $D_{\mathbf{x}}$ set to 0.95, and with the handling of simulation failures like in CRS. The value of 0.95 for p is not exactly the one giving best performance when EA is to be applied throughout the whole problem; nevertheless it was set so for two-phase algorithm to make it more explorable.

The tests were performed with varying maximum number k_{S} of $f(\cdot)$ evaluations for preliminary routines. The routines were initially stopped just immediately after the first feasible solution was found and, in next test configurations, after 500, 1000, 2000, 4000, 8000 or 16000 evaluations of $f(\cdot)$ (or earlier, if progress less than ϵ_{I} was detected). Next, COMPLEX took over. For each configuration 100 algorithm runs were made; their results are presented in Fig. 4.3. Let us discuss preliminary routines results first. Average quality of CRS solutions, presented in graphs a) and b), improves almost imperceptibly as the budget grows, while EA solutions improve steadily and noticeably. A question may be asked for the reason of evident CRS decline for big computation budget. First, consider the percentage of algorithm runs that exhausted k_{S} for subsequent testing configurations: it is $\{100, 71, 52, 46, 39, 24, 9\}$ for CRS and $\{100, 63, 54, 28, 7, 5, 0\}$ for EA. Therefore, for $k_{\mathrm{S}} \geq 2000$ EA execution is limited even less frequently by computation budget than CRS is — and EA is still able to find better solutions. A look at e) and f) graphs may bring the answer: CRS running times (in terms of number of $f(\cdot)$ evaluations) vary wildly but they are, on average, shorter than those for EA. Simply, for increasing computation budget, CRS is less and less reliable to find better and better solution in every single run — unlike EA. CRS success (cf. Table 4.1) has the price of tight termination criterion plus abundant computing budget; CRS applied for rough optimisation performs worse than EA does.

Coming to COMPLEX results, a striking observation can be made that the quality of CRS or EA solutions (subsequently taken over by COMPLEX as its starting points) affects only

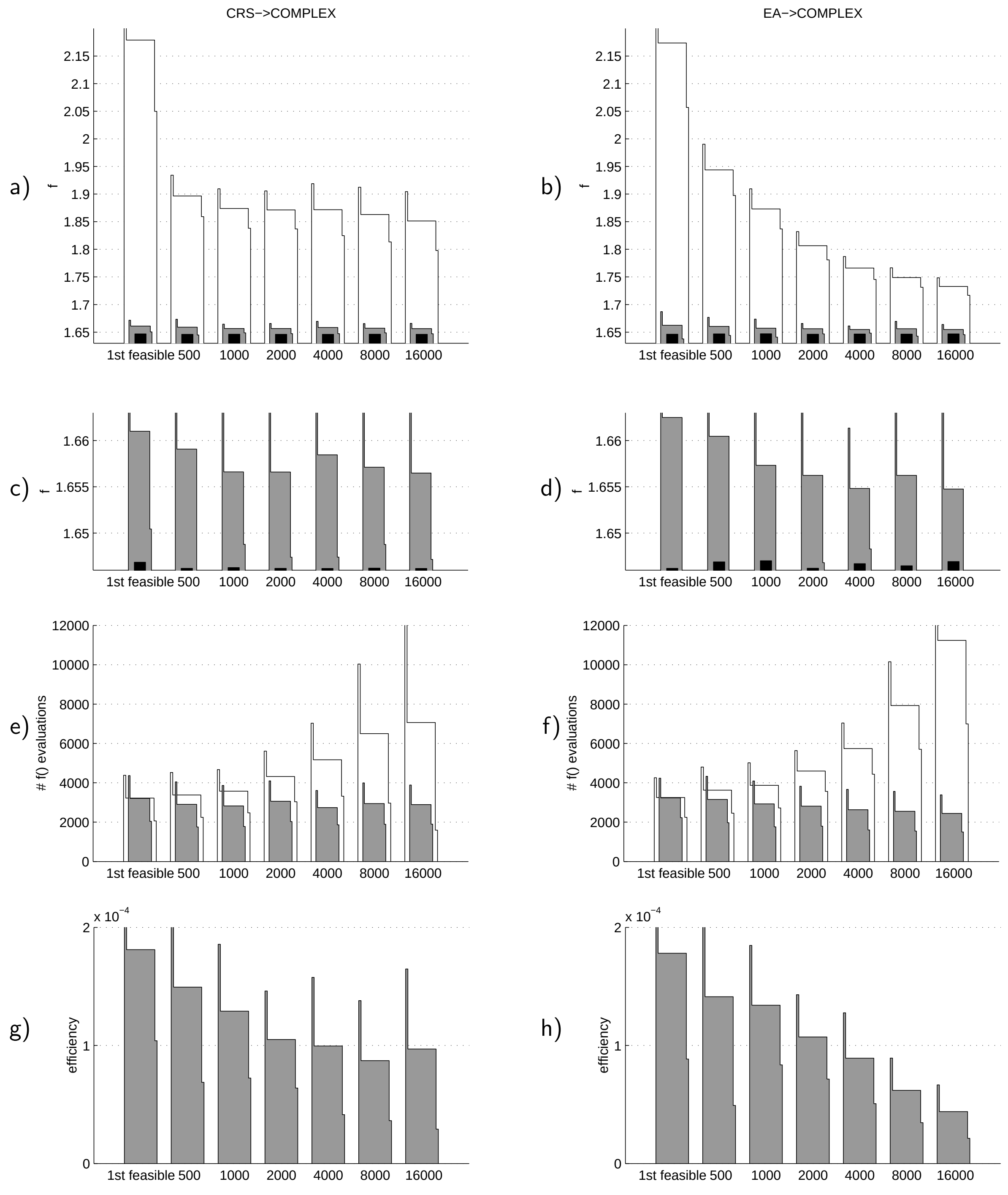


Figure 4.3: Performance index value, number of performance index evaluations and optimisation efficiency for different preliminary optimisation routines and various changeover criteria. ab) Average $f(\cdot)$ values (white bars) when the switch to COMPLEX was made; average $f(\cdot)$ values (gray bars) at the optimum found by COMPLEX; $f(\cdot)$ at best solution (black bars) found in the series of runs. cd) Close-up of the above bar graphs of final $f(\cdot)$ values. ef) Average number of $f(\cdot)$ evaluations done by COMPLEX routine (gray bars) and by the whole hybrid (white bars). gh) Average efficiency of the whole hybrid algorithm. Peaks on bar edges indicate standard deviations from the average values.

slightly the final solution quality. Compare graph a) with c) and b) with d) — and observe that correlation of white and gray bar heights can be assumed safely only in '1st feasible' case and, to some extent, for b) and d) graphs with $k_S \leq 4000$. It is evident that running either CRS or EA longer than necessary to find the first feasible solution, is pointless. This observation may mean that COMPLEX, if granted computation budget big enough, is in position to crack the problem starting from any feasible point. (This hypothesis is supported by graph d) and f) — better starting points reduce significantly the running time, not the final solution quality.)

Let us discuss the results with the efficiency as the main performance criterion. A reasonable efficiency definition in our case can be the ratio of decrease of performance index at the solution w.r.t. the first feasible point found, and the total number of evaluations of $f(\cdot)$. The graphs of efficiency defined this way are shown in Fig. 4.3gh. The evident conclusion is that switching to COMPLEX should be performed as early as possible, i.e. on finding first feasible solution. The effort spent on finding good COMPLEX starting point is useless, which is clearly seen on graph h).

When the best results (indicated by black and most embedded bars) obtained in a series of 100 algorithm repetitions are concerned, their quality — as compared to an average COMPLEX solution — makes them worth finding. Unfortunately, they are hit by COMPLEX only occasionally and, sadly to say, this can mean that the best approach could be to start COMPLEX a multitude of times from *any* feasible point. However, with the average of 2500 $f(\cdot)$ evaluations in a single COMPLEX run, such approach could be acceptable only in powerful parallel computing environment — which is not our case.

4.2 Final hybrid approach verification for the plant problem

Before getting to the target problem of full-size power plant model set-point optimisation, it is desirable to make intermediate conclusions. It comes from results of hybrid algorithm tests that, in fact, COMPLEX should be started as soon as a feasible solution is available from the preceding routine. No further CRS or EA operation is justified because of their inferior efficiency. At least such conclusions apply to our concrete problem where, as it turns out, hitting at random a point from which COMPLEX converges to $\mathbf{x}^\star$ is quite easy. One should, however, not forget that box constraints on decision variables in the test problem were adjusted so that infeasible region size was not overwhelming. Otherwise, using a preliminary routine is a must. Also, disappointingly straight rule to switch at first feasible point may be in force for this particular problem only, where non-existence of local optima is suspected. The above and all other potential criticism that can be made about the proposed approach must yield to the fundamental conclusion that the problem is solvable and the solution is satisfactory.

The main observation made so far on the switching criteria, the 'early switching imperative', can still serve as starting point for more research. The target full-size optimisation problem may serve not only to verify efficacy of the two hybrid routines, but also as an opportunity to propose and verify more algorithm amendments. Those amendments take two directions. One is to try switching criterion other than just the number of $f(\cdot)$ evaluations. The other is to make switching more effective in general, by passing as much useful information to COMPLEX routine as possible. The alternative switching criterion that was tested and whose application is described below bases on the current absolute progress (in terms of performance index improvement), i.e. the progress made within some history window. It is therefore much like ϵ_I termination condition. In detail, the changeover to COMPLEX is made only after the current efficiency η_n has decreased below some η_min. The current efficiency is calculated at, say, n-th evaluation of $f(\cdot)$ done by an algorithm as the ratio: $\eta_n = \frac{f_{n-l}-f_n}{l}$, where l is a time window, and f_n, f_{n-l} are the currently best known approximations of $\mathbf{x}^\star$ known at n-th and $n-l$-th evaluation of $f(\cdot)$, respectively. The current efficiency is calculated each time $f(\cdot)$ evaluation is invoked. Simultaneously, a pool of at most m best feasible solutions as found by an algorithm, is maintained. On $\eta_n < \eta_\mathrm{min}$ the current algorithm (either CRS or COMPLEX) is terminated, and COMPLEX is initialised with all stored best solutions as the complex vertices. Thus, the two kinds of amendments have combined into a new switching strategy. Given general conclusions from the former section, this strategy would be senseless without the improved COMPLEX initialisation. However, passing all the best information from preliminary to final routine may boost the latter so that the whole tandem could outperform the 'early switching' option.

Such switching method has been tested for the target optimisation problem. Routines settings had to be adjusted slightly in order to reflect increased dimensionality. The history window w was set uniformly for all routines to the value of 210, i.e. exactly to $10\dim\mathbf{x}$. This was a slight change in case of CRS and EA; in case of COMPLEX this adjustment was mitigated somewhat by setting $\epsilon_\mathrm{I} = 10^{-4}$ (for the test case ϵ_I was set to 10^{-5}). Moreover, EA population size was adjusted to 120 individuals, a value that does not follow the dimensionality growth (such change complies with a known hint not to make the population too large). Let it be reminded that, after what has been experienced about the behaviour of the considered routines, such settings are to detect situations when no further progress will be made (rather than logging a real optimisation progress slow-down). They are only auxiliary here.

During tests, preliminary routines (CRS, EA) were run 60 times with no initial solution

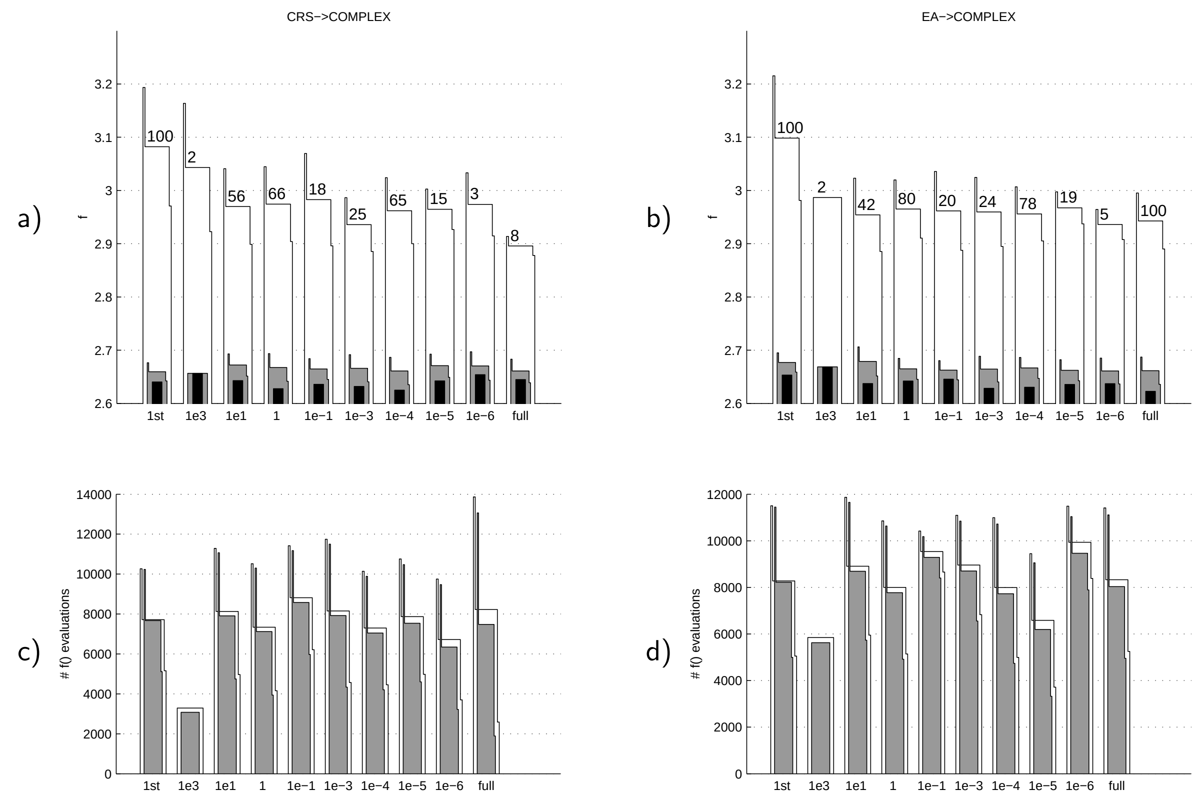
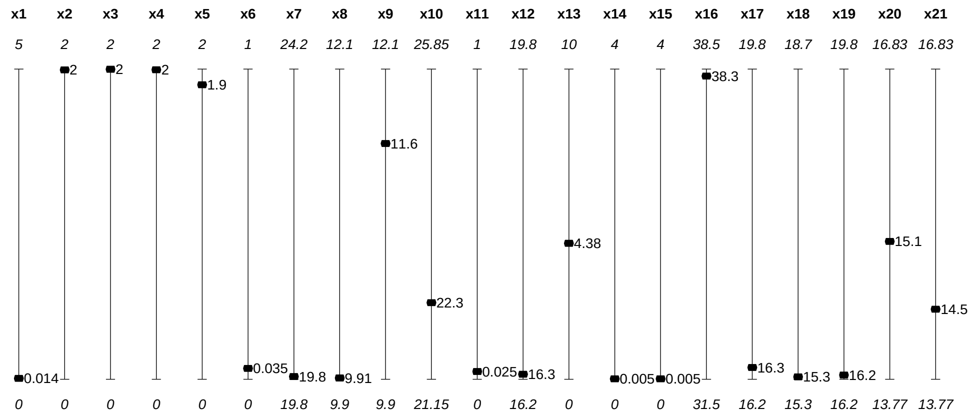


Figure 4.4: Performance index $f(\cdot)$ and number of $f(\cdot)$ evaluations for different preliminary routines and for varying switching criteria. ab) Average $f(\cdot)$ values (white bars) when the switch to COMPLEX was made, average values (gray bars) of the optima found by COMPLEX; $f(\cdot)$ at the best solution (black bars) found in a series of runs. cd) Average number of $f(\cdot)$ evaluations done by COMPLEX (gray bars) and by the whole hybrid (white bars). Numbers on top of bars in ab) graphs denote the percentage of all trial runs for which that switching criterion was reached. Labels '1st' and 'full' stand for '1st feasible' resp. 'm feasible points collected' criteria.

given. At selected milestones COMPLEX was called to finish up the optimisation task. Those milestones were: first feasible solution found, $\eta_{\min} = 10^3, \ldots, 10^{-6}$ and, independently, the moment when the pool contained, for the first time, exactly m solutions. The efficiency calculation procedure settings were $m = 2 \dim \mathbf{x}$ (i.e. exactly as many as there are in COMPLEX pool) and $l = 10 \dim \mathbf{x}$. The results are presented in Fig. 4.4. Sadly enough, all the innovations in this test series — changed switch criterion, COMPLEX supplied with extra pool points, new optimisation problem — did not bring any breakthrough; the graphs look flat and indifferent to the switching criterion. Closer examination of optimisation log files showed that only in a small fraction (about 10%) of trials the optimisation process reached the milestones harmoniously, i.e. with steady progress of gradually decreasing rate. In most cases new solutions rapidly ceased to be discovered, and the routine efficiency plunged to zero. All this is reflected by percentages of runs in which a particular milestone was reached, that are given at bar tops in Fig. 4.4ab. Particularly on graph b) those percentages rise periodically, thus marking the path most optimisation trials followed. (Results for $\eta_{\min} = 10^2$ and $\eta_{\min} = 10^{-2}$ are not shown due to lack of data.) Similarly inconstructive are statistics of the average $f(\cdot)$ evaluation number given in Fig. 4.4cd; the only conclusion is that lion's share of computation budget is consumed by COMPLEX, and that consumption can vary wildly.

Let us present the best solution found by COMPLEX routine. It is (cf. Fig. 2.2)

$$\mathbf{x} = \begin{bmatrix} 0.014 \\ 2 \\ 2 \\ 2 \\ 1.9 \\ 0.035 \\ 19.8 \\ 9.91 \\ 11.6 \\ 22.3 \\ 0.025 \\ 16.3 \\ 4.38 \\ 0.005 \\ 0.005 \\ 38.3 \\ 16.3 \\ 15.3 \\ 16.2 \\ 15.1 \\ 14.5 \end{bmatrix} \tag{4.2}$$



and the value of $f(\cdot)$ there is 2.623. The values of flows are presented in (4.2) also on rulers, analogously to (4.1). The main evaluation criteria for solution reasonability are flow distributions for boilers and turbines, and also values of controllable flows through reduction valves. Boilers 1–3 receive almost as little water as possible in order to maximally reduce utilisation of that part of installation (i.e. boilers and turbogenerators 1–3) as being less efficient. This is compensated by the maximal load put on Boiler 5 (x_{16}), the best one. As regards steam distribution across turbines, a strange situation can be observed that the flow x_9 powering less efficient TG3 is not reduced to its lower bound in order to direct more steam to more efficient TG1. The reason for this is probably the fact that TG1 output is connected to the rest of system in other way than TG2 and TG3, and may be subject to additional restrictions (implicit constraints). Instead, the way TG4 and TG5 are powered remains correlated with their efficiencies. Note that steam that reaches the last turbine 5 stage ($x_{20} + x_{21}$) is substantially bigger that the steam flow x_{10} through the less economic turbine 4. In the opinion of IHE staff, a very strong point of the obtained solution is that flows through the reduction valves are kept virtually at zero level (cf. values x_1, x_6, x_{11} and x_{15}). This means all steam is utilised for electricity generation before it reaches industrial steam outlets.

Relaxation of the plant problem constraints could possibly allow to get still better solutions. However, for the presented problem no initial solution was known, and many of constraints were set for 90% and 110% of the nominal working parameter values — just to have some problem formulation to start with. Possibly it would be wise to consider explicit constraints redefinition as soon as a feasible solution is found. However, a remarkable fact has been observed that for very 'loose' initial constraints, the search domain consists in gross part of regions where feasibility constraints are violated. This fact really justifies the use of a preliminary global algorithm of any kind instead of running COMPLEX alone.

4.3 Interface adaptation

The choice of programming environment and techniques for implementation of optimisation algorithm and for interfacing simulation has the same goals as the choice and adaptation of component optimisation routines. The first goal is to ensure stable optimisation process immune to simulator crashes; the second is to provide means to increase computing efficiency. The safest way to attain required stability is to keep simulation and optimisation routines as separate processes. (It could be also possible in most programming languages to assemble all modules into one executable; to detect the signals emitted on simulation crash, and then to intercept them — but no recipe can be given what to do next; in particular, how to reset such damaged simulator.) Therefore, after the simulator crash, the role of optimisation module is to clean the environment from all remains of that faulty simulation: hanging processes, semi-completed simulation output files etc. — and to restart the simulation module.

Unfortunately, such modules separation must result in degraded performance because simulation input and output data must cross process boundary, which requires some sort of inter-process communication to be used. Having accepted that such inter-process communication must happen anyhow, it is wise to choose such communication mechanism that allows for parallelisation and distribution of computations with small overhead.

In case of IHE modelling tool Java was chosen for implementation of optimisation module. Such choice might appear controversial, but Java is indeed apt to execute optimisation routines in cases like ours. This is because the load caused by operation of optimisation routine itself is far less than that of simulation routine — even if the optimisation routine code is interpreted. At the same time, Java has many advantages: prototyping is less error-prone, the program can be run independently of the platform, the Java class code can be just-in-time compiled before being run. But probably the biggest advantage of Java here, is its native support for parallel processing by Java threads, and the support for distributed computations with Remote Method Invocation (RMI) technology.

Some trade-off was needed as regards the degree of simulation and optimisation modules integration. On one hand, complete embedding the simulator into Java was possible thanks to Java Native Interface (JNI) and availability of simulator source code in Fortran. Such solution was, however, excluded (as mentioned above) for the sake of stability. On the other hand, it was possible to leave the simulator executable without any change — but this would result in degraded performance and impossibility of running optimisation in parallel because of simulator input and output accomplished through a set of files. It was then decided that the simulator interface was to be partially adapted — only to extent that no disk files were to be involved in transferring simulation input and output. Relatively small changes to simulation code were introduced that made the simulator read its input from the standard input (resp. write its output to the standard output), instead of using files. This sped up the simulation and, what is more important, allowed several simulation module instances to be run in the same directory e.g. in case of parallel optimisation on a multiprocessor machine.

Power systems optimisation parallelisation was eventually not practised, but appropriate Java classes and interfaces have been prepared that would make computations distribution easy. Preparing parallel versions of some of presented optimisation routines is similarly easy as well from conceptual (cf. Section 5.1 and Section B.1) as from technical (Java threads and monitors) point of view.

4.4 Conclusions on proposed solving approach applicability and efficacy

It will be shown that the way the plant set-point optimisation problem was approached fully conforms to Thesis 2, further specified on p. 82 as the proposed model approach to simulation-optimisation problems. The proposed approach steps relate to the procedure followed and presented in this chapter as follows.

Ad 1. It has been postulated that the selection of hybrid algorithm component routines is determined by any *a priori* problem knowledge, by routines simplicity that allows for eventual modifications, by routines ability to support the problem specifics in any stage of optimisation, and by routines and computing environment adaptability for concurrent execution. The *a priori* knowledge is the awareness of high model nonlinearity plus experience in techniques used by commercial (or, at least, similar) optimisation systems. To ensure possibility of modifications, the chosen optimisation routines are either simple or available in their source code. Their support for wide classes of problems minimises the risk that they will be unfit for the considered problem. Each of the selected routines can be subject to coarse-grain, however unrefined, parallelisation. Also, the selected environment is well suited for parallel and distributed computing — this coming at negligible cost (i.e., slow execution of an optimisation routine). All things considered, CRS and — originally — GRG have been chosen to be run in Java.

Ad 2. It has been postulated to make a simpler, and therefore not so computationally demanding, model — and to use it throughout the procedure in place of the original one. This recommendation has been followed by introduction, in Section 2.1, of nine-dimensional test model, which was subsequently demonstrated to exhibit all the features of the original model: all sorts of constraints, simulation 'noise', sudden steps on $f(\cdot)$ surface.

Ad 3. Appropriate measures have been taken to ensure mutual insulation of simulation and optimisation modules. Simulation interface definition has been defined so that the optimisation routine could be left unaware of eventual simulation failures. Moreover, easiness of RMI mechanism application keeps software portability to distributed version low-cost.

Ad 4. The primal requirement to carry out optimisation in order to achieve a good solution, a reference point, has been achieved. In presence of simulation failures the optimisation is possible by change made to CRS code so that those failures can be supported. On the other hand, the mounting problems in gradient estimation forced to replace GRG with COMPLEX. Consequently and similarly to CRS, COMPLEX was subject to modifications in order to make it immune to simulation failures and domain non-convexity. Given CRS inefficiency, EA has been considered as alternative preliminary routine. All these activities follow recommendations given in Point 4.

Ad 5. As suggested, series of optimisation tests have been performed with hybrid algorithms set for various switching criteria. The highest efficiency was obtained when the switching was performed as soon as a suitable COMPLEX starting point was available. The hybrid methods proved to be robust but — due to COMPLEX nature — there are advised to be run in several repetitions as it improves the chance to find a really good solution.

Ad 6. By following the last suggestion and applying the hybrid to the original full-size problem, good optimisation results have been obtained in terms of $f(\mathbf{x}^{\star})$ alone as well as in terms of analysis of the solution nature (i.e. flows distributions and load of system components).

The above are the main conclusions. Also, as by-product, experience concerning switching criteria and exact workings of considered routines has been collected. This experience will be conveyed to two other problems whose solution descriptions follow in the next chapter.

Chapter 5

Solutions of waveguide design and optimal pricing problems

This chapter presents procedures applied for solving the two other considered problems, the waveguide design and network services pricing. They both were being approached under circumstances different than those for plant problem, with differences considering mostly modeller availability or (as in waveguide design) or model variability (as in services pricing). Consequently, the applied solving procedures do not follow closely the approach postulated on p. 82. As compensation, the two cases offer an opportunity of insight into kinds of troubles one may run into that are different from simulator fallibility — but that are equally important.

5.1 Waveguide dimensions optimisation

The problem of waveguide dimension optimisation was being approached by the author in different circumstances than the power system models. The grounds of microwave models optimisation were covered in IR quite profoundly — numerous publications had been already issued from there [49, 25], reporting the actual methodology with *QuickWave* as a tool, as well as successful optimisation attempts for circuits modelled otherwise than with *QW-Simulator*. In one of them [8] the problem of a microwave transformer design was presented, which was solved by combined application of EA and a local search procedure with search direction being derived from QP.

The hopes and positive attitude in IR towards direct search methods mingled with anxiety about their well-known computational burden: solving a moderate-dimension optimisation problem with a model calculated numerically could easily last days. General-purpose evolutionary algorithms, so frequently applied to various design problems and famous for their efficacy, were particularly unwelcome in our problem. Let us remind here that the simulation times for power system models were in the order of one second while an average *QW-Simulator* run takes minutes. Therefore, the question was what routine to take for preliminary optimisation.

In the alternative, widely known and competitive to *QW-3D*, approach [13], constructed simulators are highly specialised to handle a narrow class of designed shapes. Properties of performance index computed this way let the authors successfully utilise gradient methods to find the optimal design. It appears that the crucial point of this approach is a mapping of some kind between the decision variables for a coarse model and the decision variables of the 'true', fine model. Such mapping allows to perform coarse-model optimisation, with occasional referring to fine-model simulation results.

Algorithm selection

CRS2 has been qualified as the preliminary optimisation routine. Such decision has been made jointly with IR as the result of rather positive impression CRS has left after the power models case; it has been also driven by some disbelief that the application of EA in this particular case would pay off. This decision is definitely subjective since the merits of EA are unquestionable: it is as simple in implementation and robust as CRS is, and so far it has proven to be even more effective than CRS. The only EA feature that could be perceived as drawback is that some algorithm termination criteria process all individuals in the current generation and therefore termination criterion evaluation can be invoked much less frequently than for CRS.

The possibility of running the optimisation in parallel is an issue as important as robustness and efficacy. The overall circuit design process cannot exceed several days, and this condition might easily be not fulfilled considered performance properties of so far used Powell routine, and the fact that Powell is available only in sequential version. Instead, CRS parallelisation is easy, also from technical point of view, especially that the unit being at disposition is a multiprocessor PC, and not a distributed heterogenous network.

Algorithms adaptations and 2-D optimisation tests

CRS routine does not require any special arrangements to be applied to our problem. Unlike for power system modeller, *QW-Simulator* does not fail in the way that could affect stability of optimisation module execution. 'Failures' of *QW-Simulator* mean rather enormous errors that could appear in effect of improper settings — like mesh point locations, excitation wave parameters, or simulation time — but none of them is available to optimisation routine as the decision variable.[1] The effort of adaptation consists mainly in developing a CRS parallel version. The way CRS has been parallelised differs from what is proposed by its author in [92]. The concurrent CRS algorithm described there was designed to be run asynchronously in a specific transputer architecture, and consisted of a pool management process and k processors executing CRS core operations (simplex creation, feasibility tests, function evaluation). CRS processors constantly executed the core operations and communicated $\tilde{\mathbf{x}}$'s better than $\mathbf{x}_h$ to the pool manager. On the other hand, the pool manager collected those new pool points, compared them with the current pool contents, and broadcasted any changes of the pool to CRS processors.

Such implementation was highly asynchronous. In our case CRS is to run on a single multiprocessor PC, and therefore different parallelisation scheme has been proposed that assumes closer dependence of CRS processes. Point pool, instead of being managed by a dedicated process, is made a shared resource with access reserved for one CRS process at a time. A CRS process task executes without modifications with the exception that points being actually reflected by other CRS processes cannot be used for simplex creation (Step 2). Such a parallel CRS version is simpler to implement and introduces no inter-process communication delays.

Yet other two changes have been made at some point in CRS application to waveguide design process, in pursue of quicker convergence. First, it was verified empirically that reflecting $\mathbf{x}_h$ instead of some randomly chosen $\tilde{\mathbf{x}}_{\dim \mathbf{x}+1}$ really improved CRS performance. With this rule incorporated into the algorithm, $\mathbf{x}_h$ is inevitably used for reflection — and becomes blocked for other CRS processes. Consequently, other CRS processes use the second, third, and so on, worst points for reflection. Such coordinated modifications really help to get rid of worst simplex vertices quickly. The second change concerned handling of $\tilde{\mathbf{x}}$ violating explicit constraints (Step 4). Instead of rejecting them (as was done originally), they are 'bounced' at constraints back into $D_{\mathbf{x}}$

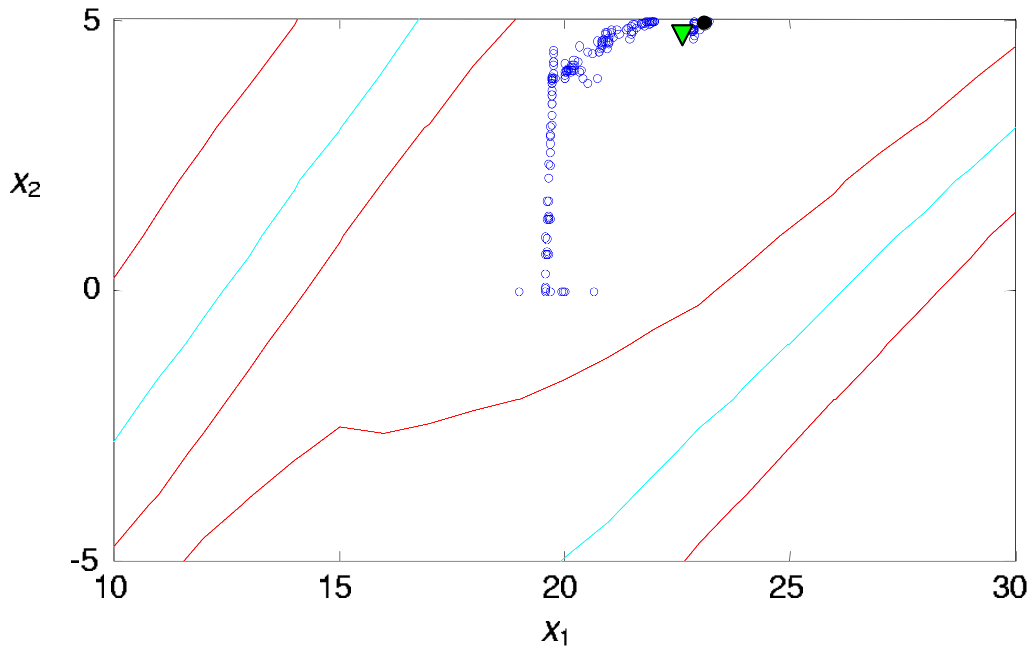
$$\tilde{x}_i := \begin{cases} 2x_{\mathrm{L},i} - \tilde{x}_i & \text{if} \quad \tilde{x}_i < x_{\mathrm{L},i} \\ 2x_{\mathrm{U},i} - \tilde{x}_i & \text{if} \quad \tilde{x}_i > x_{\mathrm{U},i} \\ \tilde{x}_i & \text{else} \end{cases} . \tag{5.1}$$

A series of optimisation runs in different configurations have been made to find out whether CRS was really capable of replacing Powell method altogether in waveguide design problem. Two of exemplary 'optimisation paths', i.e. plots of trial points generated by Powell[2] and CRS

[1]Stability of the simulation-optimisation software as a whole does not free the optimisation routine designer from being cautious since huge undetected simulation errors may pass unnoticed and may do more harm to the optimisation process than evident simulation crashes.

[2]Although Powell routine generates subsequent solutions using performance function (2.5) for a finite p,

a)



b)

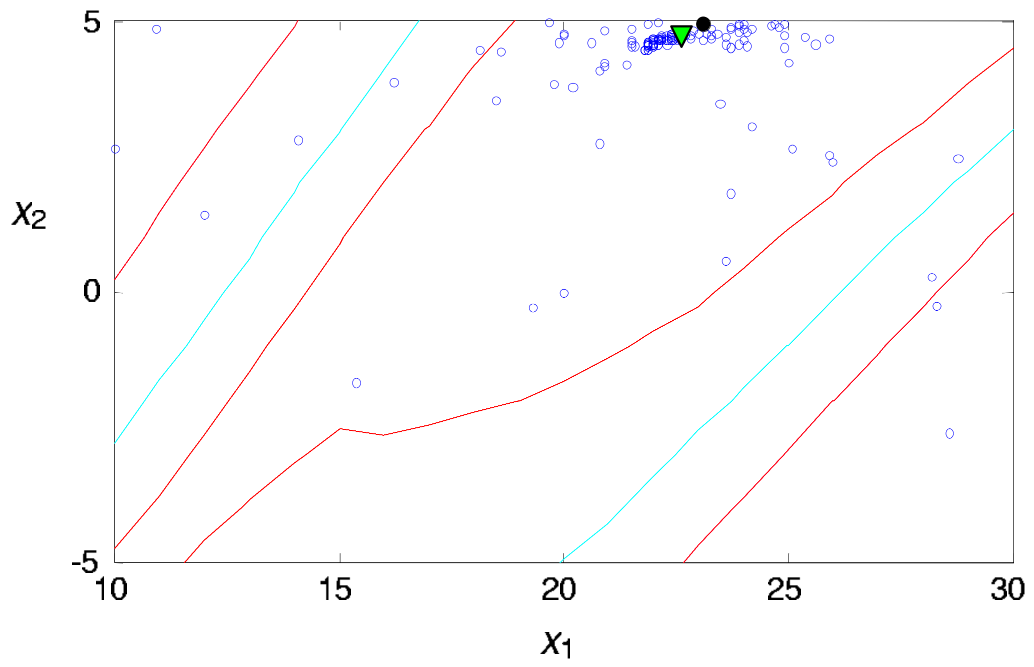


Figure 5.1: Trial points (white circles) generated in exemplary runs of a) Powell routine and b) CRS routine. The problem solution location is indicated with a triangle.

methods are presented in Fig. 5.1. The path of Powell method presented there is a specimen in the sense that it actually presents much more than Powell method experiences in its average run. One can distinctly see the first two line searches performed in versor directions, followed by searches in directions considered by the method to be conjugate. However, frequent backtracks denoted by many overlapping circles indicate that the quadratic approximation of $f(\cdot)$ definitely does not hold. This Powell path is eventually a successful one, but in general the algorithm efficacy is very sensitive to the start point location and demands attentive supervision of an expert who is needed to apply restarts and to adjust p in (2.6). Instead, the other graph presents an ordinary CRS path. It was experienced that using CRS one needs on average 30% less budget for performance index evaluations than while using Powell routine, to achieve a solution of comparable quality. Such ratio is a resultant of average number of $f(\cdot)$ evaluations performed by each routine (much lesser in case of Powell) and the average efficacy of a single optimisation run (much bigger for CRS). Forty optimisation runs of each routine were performed to obtain those ratios, with the starting point being chosen at random in each case.

Fig. 5.2 presents location of 'good' CRS solutions and the location of the best solution found by Powell routine against the location of the problem solution $\mathbf{x}^\star$. 38 out of 40 CRS solutions have been classified as as good w.r.t. the value of the objective function; however some of them lie in the attraction area of the local optimum. Great majority (circles tend to cover one another) of CRS solutions are located very close to true problem solution,[3] which is inaccessible because of simulation errors forming a 'canyon wall', against which most solutions lean. (This canyon wall could be reduced by setting appropriate simulation accuracy — but this would cause a single optimisation run to last days.)

3-D optimisation by a hybrid routine

The results of two-dimensional waveguide optimisation example give CRS absolute primacy over Powell routine. It can seem the cooperation of them both algorithms in any form of a hybrid algorithm would be pointless. To verify this observation, a number of optimisation trials has been performed in similar configuration for 3-D problem (i.e. with *ims* as the extra decision variable x_3). Powell and CRS were started from points randomly selected from the domain. Out of 100 optimisation runs, the number of successful ones, i.e. with performance

its termination criteria use $\|\cdot\|_{L_\infty}$ — i.e. as in CRS. This is another Powell routine improvement aimed to prevent the premature convergence.

[3]The true problem solution was found by extensive scan of the most promising area (the one densely populated in Fig. 5.2 with circles). The simulation settings used for the scan ensured very high accuracy.

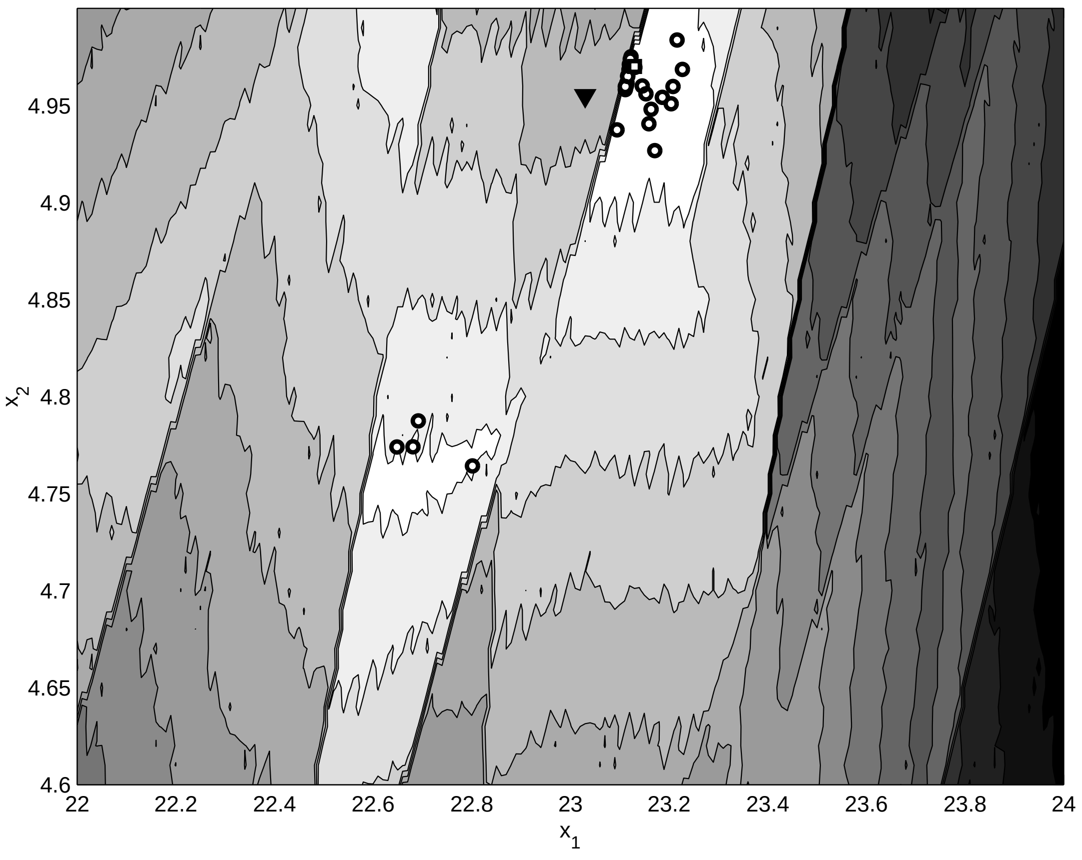


Figure 5.2: Location of solutions found by CRS (white circles), of the best solution found by Powell routine (white square) and of the true solution (black triangle) for two-dimensional problem of waveguide design.

index value reduced to below 0.012, was 8 for Powell and 49 for CRS routine. Simultaneously, the average number of $f(\cdot)$ evaluations needed by the algorithms to reach a solution was 34 and 260, respectively. Let us introduce some measure of efficiency: let us call it the hypothetic effort needed to be put in order to find a satisfactory solution with probability p:

$$N = n \log_{1-r}(1-p) \ , \tag{5.2}$$

where n is the average number of $f(\cdot)$ evaluations made by an algorithm and r is the ratio of successful solutions obtained in a series of optimisation runs.[4] Let us require the probability p of obtaining a good solution to be 0.95 and let n denote the average number of $f(\cdot)$ evaluations needed to find a solution. Then, for the experiment results given above, such hypothetic effort would amount to $34 \log_{0.92} 0.05 \approx 1222$ and $260 \log_{0.51} 0.05 \approx 1157$ evaluations of $f(\cdot)$ for Powell and CRS, respectively. It is plainly seen that CRS still wins over Powell, although the fraction of good CRS solutions is considerably less than for the 2-D case.

Basing our efficiency measure on the number of $f(\cdot)$ evaluations after which a routine finds its solution, can be criticised. Possibly it is better to let n denote the average number of $f(\cdot)$ evaluations done by an algorithm before it is stopped. So defined effort in our experiment is equal to 167 for Powell and 1000 for CRS. These numbers are relatively high and indicate that the termination criteria defined by IR staff are quite tight for both routines; however, by applying the same measure of efficiency, CRS ($1000 \log_{0.51} 0.05 \approx 4449$) is still placed well ahead of Powell ($167 \log_{0.92} 0.05 \approx 6000$). Undoubtedly, so specified termination criteria result from former IR experience that a new, better solution is likely to be discovered even after a lengthy period of unsuccessful trials. This complies absolutely with the nature of CRS, which needs time to replace its pool contents. It also fits the nature of modified Powell routine, which often discards would-be new solutions if they are not optimal w.r.t. $f(\cdot)$ defined for $\|\cdot\|L_\infty$ norm.

Given the multiminima nature of $f(\cdot)$ and decent Powell routine convergence for some good starting point, one has the right to suspect that combining CRS with Powell in a hybrid method could attain better efficiency than its component routines separately achieve. There is, however, the usual issue about switching criterion selection. From what turned out in power plant case it can be judged that unconditional switching after some k_S iterations have been completed (cf. p. 87) is more promising. To verify that, fifty repetitions of 3-D problem optimisation have been performed using a hybrid algorithm with CRS being the preliminary

[4]Formula (5.2) results from the following reasoning: if $1-r$ estimates probability of the event 'an unsatisfactory solution is found in a single algorithm run' then we have to perform $\log_{1-r}(1-p)$ optimisation trials in order to reduce that probability to the value of $1-p$. Since a single optimisation trial comes at cost of n performance index evaluations, and (5.2) follows.

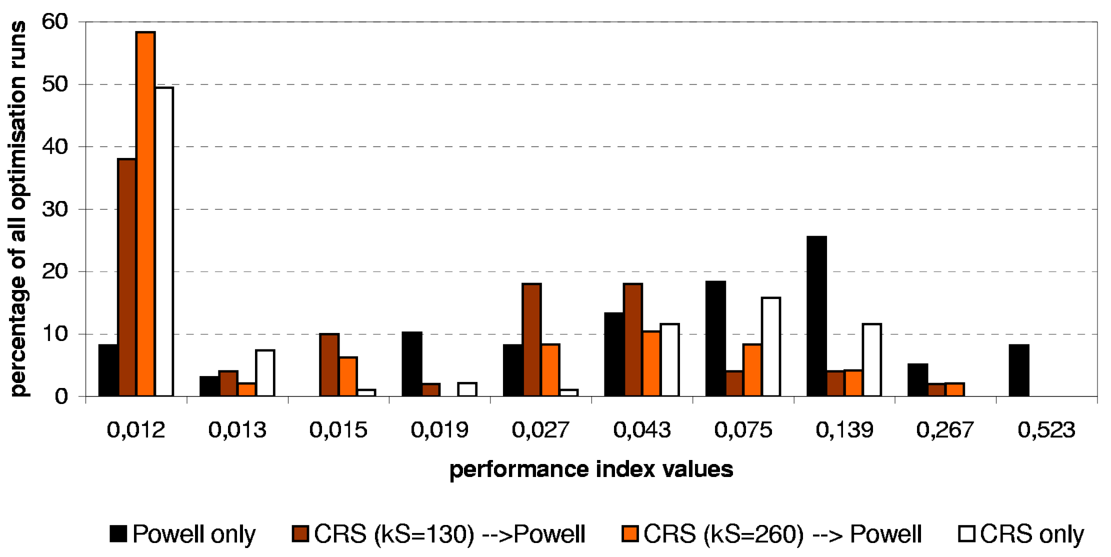


Figure 5.3: Histogram of performance index values for solutions found by Powell (black bars) and CRS (white bars) routines alone as well as for a hybrid made out of them with the switching from CRS to Powell made after $k_S = 130$ (dark gray bars) and $k_S = 260$ (light gray bars) performance index evaluations. Numbers below bars denote interval upper values.

and Powell the final routines. The histogram of $f(\cdot)$ values found by the hybrid for two k_S values is presented in Fig. 5.3. For convenience reasons, the same graph contains results of optimisation with only CRS and Powell routines, discussed above. The noticeable fact is that $f(\cdot)$ values cluster in two intervals, $< 0, 0.012 >$ and, more frequently, $< 0.019, 0.267 >$. The latter one contains solutions that are not considered as interesting; they are located either in the local optimum attraction area, or in the area around the global minimum — but for some reason that minimum proximity has not been reached. From our point of view such solutions are useless. Our attention is focused on those good solutions (wherever they lie) with performance index less than 0.012. It turns out that the hybrid algorithm, in its both versions, is clearly much more effective than Powell routine applied alone. Moreover, the version with $k_S = 260$ outperforms also CRS run alone. This could mean that Powell, started from a good point, is able, in some circumstances, be more effective in finding a good solution than the much praised CRS.

Let us see now how the efficacy in the considered interval relates to efficiency calculated using (5.2). Table 5.1 gives the ranking calculated for the average number of actual $f(\cdot)$ evaluations done by each of the tested algorithms. It turns out that by applying a hybrid routine one can significantly reduce the effort needed to obtain a good solution with acceptable confidence.

In another series of optimisation trials efficiency-based switching criterion was tested that

	results\algorithm	Powell only	hybrid ($k_S = 130$)	hybrid ($k_S = 260$)	CRS only
1.	average number of $f(\cdot)$ evaluations	167	223	338	1000
2.	percentage of good solutions	8	38	58	49
3.	efficiency	6000	1397	1167	4449

Table 5.1: Statistics related to efficiency and efficacy of 3-D waveguide problem solving by various algorithms (Powell routine, hybrid algorithm with CRS/Powell switching and diverse termination criterion, CRS routine). 1. Average number of performance index evaluations calculated for those algorithm runs in which $f(\cdot) < 0.012$ was obtained as result. 2. Percentage of those algorithm runs in which $f(\cdot) < 0.012$ was obtained. 3. Efficiency estimate calculated using (5.2), for the actual number of iterations made.

was exactly as the one presented in Section 4.2 but, similarly for IHE case, the results were not impressive. The starting points produced by such switching algorithm in a single run, that should mark milestones of decreasing efficiency, were all the same. Only in about 15% of all simulation trials those milestone points are properly diversified. For comparison, the ratio for k_S-based switching rule was about 40%. Therefore, application of efficiency-based switching rule has been discarded.

There is another conclusion that is interesting from the engineer's point of view. Looking at 2-D and 3-D optima locations (Fig. 5.4), one can see two things. The first is that in each case they form nearly a line — it means that the proportions of the waveguide bend have to be preserved in order to let the wave pass easily. The second is that 2-D and 3-D optima lie quite apart. This is due to x_3 variable, in 2-D case kept frozen at the value of 6.0, and now let free. Since the optimal x_3 value is consequently set very low (in the order of tenths) by both CRS and Powell routines, x_1 and x_2 must be decreased accordingly — again, to preserve the bend proportions. This seemingly unimportant detail indicates that a surplus of decisions variables leading to continua of equally optimal solutions is not uncommon in practical automated design problems.

As it was reported in [72], the main drawback of CRS is its efficacy that drops rapidly. We had chance to observe such efficacy decrease while including x_3 to the vector of decision variables. The decrease proceeds very fast so that solving an 8-dimensional design problem with CRS alone is out of question. However, application of the hybrid routine in such cases

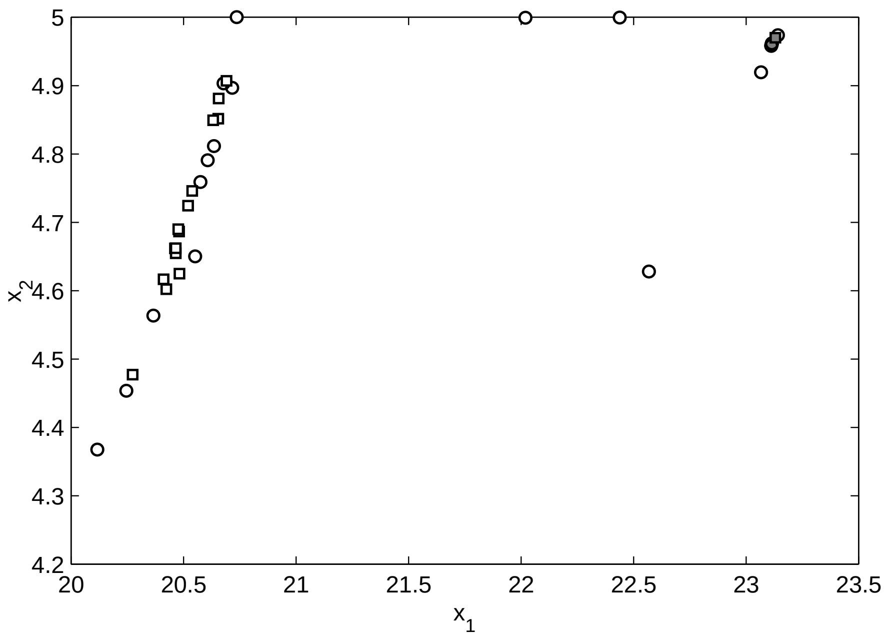


Figure 5.4: Location of results obtained for 3-D problem optimisation (white marks) and for 2-D problem optimisation (gray marks). Circles denote Powell solutions, and squares — CRS solutions.

has not been tested yet.

Interface adaptation

It must be stressed that all numerical experiments involving *QW-3D* software were performed with kind approval and close supervision of the software manufacturer, QWED, Ltd. Many of those experiments, designed jointly, have been carried out exclusively by a colleague from IR. Such limited freedom for development and for testing of optimisation algorithms undoubtedly had to affect also the choice of simulation-optimisation interface. (The limited freedom has also its advantages, allowing to shift much research, testing and result interpretation effort into the hands of a specialist and practitioner).

It was a precondition of cooperation that new optimisation algorithms must have been integrated into *QW-Optimizer*. This has determined the programming language to be C++, and the operation system to be *MS-Windows*. Furthermore, every new optimisation routine, apart from the requirement for parallel implementation, had to be run in 'quants', periodically giving up control to the system — so that it could fit the standard *MS-Windows* message dispatch loop.

Parallel CRS implementation has followed the above guidelines. The parallel CRS processes have been implemented as RWC[5] threads; the critical section of point pool access has been accomplished by using appropriate RWC mutex. Therefore, CRS can be executed concurrently in its threads, undisturbed by the requirement to give up control periodically. It should be also mentioned that, since *QW-Simulator* interface is, in fact, files, temporary directories had to be created and removed automatically for each CRS thread, to protect against mutual interference of *QW-Simulator* instances.

It should be noted that, since *QW-Simulator* is interfaced with files, it was possible to write a separate optimisation module executable, like in case of IHE simulator. However, it must be emphasised that writing a simulator-optimiser interface is usually quite a big but hidden programming burden, and that obeying implementation rules imposed by *QW-Optimizer* structure in order to avoid such effort was the easier and safer way.

5.2 IP services price optimisation

The problem of IP services pricing was chronologically the last one approached by the author. Naturally enough, that approach was biased by the experience of problems discussed above. The bias consisted in expecting similar difficulties to those already encountered: unconnected domains, nonconvex performance index, unpredictable disturbances born by simulation process. Consequently, such difficulties were found, one by one, and were presented in Section 2.3.[6] There was no difficulty in finding difficulties common to all simulation-optimisation problems, or rather to problems where switching and advanced numerics are in the play. So there was no difficulty in pointing out optimisation methods, or their combinations, able to cope with them. This is why those expected optimisation problems will be only mentioned here shortly in the context of the simple self-made model known from Section 2.3. What is more interesting to be discussed here is what a completely different problem can be generated by the same modeller, if given into hands of a person with slightly different background.

[5]Rogue Wave Class (RWC) library provides a programmer with an abundant set of classes that are fundamental for development (e.g. string, date, thread) and simultaneously hide most of a concrete operating system specifics used to implement their functionalities. RWC represents a line of products [98] that make it possible to program applications at certain abstraction level.

[6]The market model presented in this thesis has been deliberately deprived of functionalities whose behaviour may have effect similar to numerical errors. Such functionalities are deliberate rounding of particular internal model variables and numerical integration of some distributions that are nonparametric but have their origin in traffic statistics. Some of their potential effects are discussed in [60].

Self-made difficulties are easiest to solve

As it was already mentioned, IP market model and the optimal pricing approach have their predecessors in other products by one QOSIPS partner. Specifically, modified SLP routine (presented in its pure version on p. 66) was usually employed for optimal price calculation. However, formerly used models were either linear or easy to linearise, or gradients could safely be estimated numerically. In case of PM there were reasons to expect analytical gradient calculation to be impossible, and its estimation to be unreliable. This was due mainly to heavy internal discontinuities that took place in tariffs for network utilisation. The switching resembled somewhat current mobile telephony tariffication schemes, usually with big initial charge when the transmission starts, followed by finer discretised charging.

Above all, justified anxiety dominated that the optimisation problem may turn out to be non-convex and the modified[7] SLP (called from now on SLR), being a local method, will fail. Following the much frequented way, SLR was put into contest with CRS and COMPLEX to solve the problem presented in Section 2.3.

Contour plot of the performance index (consisting mainly of utilisation-based income), feasible set boundaries and location of solutions are given in Fig. 5.5. SLR was started four times from four starting points $[x_1, x_2]^T$, $x_1, x_2 \in \{0.3, 0.5\}$, with support for implicit constraints on the number of customers accomplished through a penalty function. CRS and COMPLEX were started 20 times from $\mathbf{x}^T = [0.3, 0.3]$, i.e. from the corner opposite to where the solution was. The two routines were used in the same form as presented in Chapter 4.

It is plainly visible that SLR, run with its original (i.e. adjusted for previous models) parameters, does not work. Saying that we do not mean attaining the global optimum, which SLR is not supposed to do, but simply getting closer to boundaries where penalty gets activated. Only after — it must be admitted — minor modifications SLR, started from $\mathbf{x}^T = [0.5, 0.5]$ and with big computation budget, yields results comparable with that of CRS. The case of SLR only supports the thesis that closed and specialised software products like the one SLR was taken from, integrate quite well modelling and simulation routines. Location of CRS and COMPLEX solutions comply with their general reputation and numerical results obtained for the other models. COMPLEX easily detects that the solution must activate a constraint, but it supports unconnected domains not that well as CRS does (note the three COMPLEX solutions remaining in worse subdomains). Instead, CRS pays for its globality

[7]The modification of the original SLP method consists in that the solution of a problem linearised around the current solution is not adopted as a new current solution. Instead, some point between the current and the new solution is adopted. That is why the method is called Sequential Linearisation with Relaxation. The point of this seemingly simple modification is a good heuristics for how much the new solution should be relaxed.

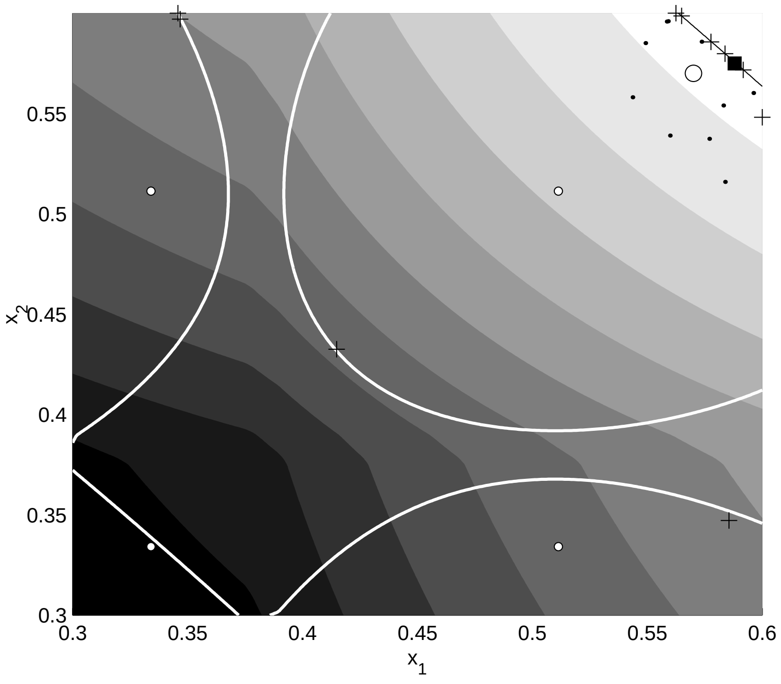


Figure 5.5: Performance index contour plot for 2-D services pricing problem, with four feasible regions bounded by the graph border or by black and white lines (cf. Fig. 2.17). Brighter areas denote better performance. CRS solutions are marked with dots; COMPLEX solutions are marked with crosses; the best SLR result and SLR starting points are marked with big and small circles, respectively. The problem solution $\mathbf{x}^{\star}$ is marked with a black square.

with imperfect solution quality.

Apparently, the conclusion could be that application of CRS/COMPLEX hybrid again (or some other routines of equivalent properties) would do for this problem. However, no results of numerical experiments will follow here, since a problem quite different quantitatively and qualitatively was prepared by one of QOSIPS partners.

Unexpected simplifications can puzzle

It must be stressed that IP services market model developed and incorporated in PM was exactly what was required and agreed upon by all QOSIPS parties. Nevertheless, the partner that was to use PM, found the first release useless and, basically, not working. The reason was the same as emphasised in Section 2.3: PM, by its flexibility, allows to define diametrically different optimisation problems by changing such superficial parameters as bounds on $\mathbf{x}$ or penalisation schemes for QoS failures. The model makers, as well as the rest of QOSIPS consortium did not feel it necessary, and sometimes legally possible, to acquaint modeller R&D team with day-to-day price setting practice of end user marketing team. Meanwhile, those marketing realities in fact determine the pricing problem features.

The details are as follows:[8]

- Apart from being limited by model validity, the scope of decisions an NSP can make is constrained by policy of other NSP's that dominate the market. Despite the optimisation problem was always being presented as a continuous one, the price decisions made by humans were two-valued: either price something exactly as leading NSP's do, or apply some discount; try to make out which discounts are most appreciated by customers and have least impact on the profit.

- SLA's currently made by NSP with customers are loosely related to actual QoS in the network — therefore the free component of our dynamic system response is very weak, and the system itself becomes much simplified.

- Instead of several key prices, tens of decision variables are declared in the optimisation problem formulation. Relations between those variables are next controlled by a multitude of equality constraints (made out of inequality constraints) that render the feasible set zero-measure.

One can try to find reasons for the above circumstances. Possibly, all of them result from human imperfections, ignorance or conservativeness existent in all phases of model design,

[8]Some of those remarks and conclusions have been already reported in [10].

implementation, testing, deployment — and finally — staff training. However, like the faulty simulator, they are perceived in this dissertation rather as pre-existent situation one has to act upon.

The above circumstances have very serious implications. First, the model so limited virtually behaves as linear. It loses all attributes that made CRS and COMPLEX the favourite optimisation routines. Especially the last fact, the introduction of equality constraints, much degrades performance of CRS applied alone, and disqualifies COMPLEX altogether. To say the truth, all the above circumstances make the optimisation task simpler, provided one knows them in advance.

In such situation SLR was obviously brought back to service, and has dealt with the problem perfectly. The concluding remark can then be made that one should approach unknown problem with a set of various optimisation methods that rather complement each other than compete each other.

5.3 Conclusions on suggested approach to optimisation problems of unknown nature

Following what has been declared at the beginning, elements of the proposed solving approach (cf. p. 82) will be pointed out for the problems presented in this chapter. As regards Point 1, the routines selected initially (CRS and Powell, CRS and COMPLEX) cover the optimisation in global and local stages, remain simple to modifications and (except Powell) are easily parallelisable. Their selection is affected by previous experience about their nature and about the nature of the problem. As regards Point 2, both problems have been initially prepared and solved in their simpler versions (2-D waveguide model, 2-D IP market model). The interface design (Point 3) has been completely determined by external factors so that no freedom in its definition was left to the author. However, support for parallel execution was the requirement imposed in case of waveguide design problem. As far as Point 4 is concerned, the changes that turned out to be necessary to solve simplified problems were far lighter than in power system case; they appeared only in waveguide problem case. (They consisted in modifications of Powell routine termination criterion, and in improved reflection operation in CRS.) The suggestion contained in Point 5 to work out an efficient and effective procedure has been followed in waveguide problem case, where switching on exhaustion of computation budget by preliminary routine turned out to be most reliable and efficient procedure. As for IP services pricing problem, neither this nor the next, Point 6, was applied because of complete model redefinition by the customer.

Perhaps numerous results of numerical experiments given in this chapter and in the preceding one have obscured the major goal of simulation-optimisation software end user. The chase of best parameter values for optimisation routines and most efficient switching criteria cannot override what is the most important: the user usually wants a satisfactory solution — not a global one neither the exact one — just a good solution improving the current situation considerably. The fact that in the examples presented here a 'global' routine (CRS, EA) was run first did not mean that finding the global solution was the prime goal. First, used algorithms are merely heuristics and, despite their global nature, do not guarantee the global solution to be found as does e.g. Meewella-Mayne method [69]. The major point in their application is to find a decent starting point for another, more efficient method to be run as the next one. Similarly, running that method with tight termination criteria that resulted in solution accuracy impossible to obtain in the real life was done deliberately to find out the nature of the solution (e.g. whether it lied on constraints). In practice, efficacy and robustness of the optimisation routine are the main objectives; the efficiency can be, to some extent, attained by more powerful computer hardware and parallelisation.

The approach presented here is located between the most advised but often not feasible strategy of profound preliminary examination of the underlying model, and the quite often practised strategy of 'brute force', which skips over any model information to apply some sort of complete scan of solutions. The hybrid algorithms presented here are only heuristics and, although backed by scores of successful practical applications, do not guarantee the solution of any problem — and therefore Thesis 2 is not proved. To compensate the loss, another thing is guaranteed — namely, that following the procedure proposed on p. 82 one gets involved in the problem specifics as little as possible, only as much as it is necessary to find some satisfactory solution. The question may be asked, *why* actually to prevent someone from getting deeper into the model — especially that many books suggest to do the opposite? It is because profound recognition of model nature is, of course in particular cases, impossible or costly in sense of human effort. Still more difficult is changing the existing model behaviour. For example, in case of IHE modeller, the core numerical routines are the heritage of many years' work of other specialists and, at least in the period of cooperation with IHE, there were apparently no human resources to be put at work to improve the model. In case of *QW-3D* the simulator is a commercial product, and any change of its behaviour is impossible. The only approach that guaranteed the author the minimum autonomy of work in that case, was to carry out 'behavioral studies', i.e. to rely on the model output only.

Let us now argument shortly the subsequent points of Thesis 2 with the experience presented in this one and the previous chapters. As regards Thesis 2a suggesting the appointment of component routines for a hybrid algorithm, the following can be stated. The candidate algorithms should constitute a team capable of carrying out the optimisation process in all phases. It seems that a global routine, regardless whether CRS or EA or some other one, is needed for all considered optimisation problems. Next, the local routine depends on the problem, but in general either direct search or gradient algorithm is executed. As regards Thesis 2b suggesting making *ad hoc* routines modifications, those modifications should be applied only in order make a workaround for a particular difficulty. They should not be re-used blindly for other optimisation problems; they rather ought to be introduced or developed only on demand. As far as Thesis 2c is concerned, it is difficult to establish uniform criteria for switching between the algorithms. Here, only efficiency-based switching rules have been considered and, regardless of particular results, the efficiency should remain as one of those criteria in one aspect: prolonging lack of progress must result in running another routine. A promising but demanding switching criterion could be the change of problem nature — e.g. detecting at some point that the problem can be linearised. However, for the examples considered such criterion has not been tested. The postulate in Thesis 2d to speed up the calculations by running them in parallel has as its aim improving the efficiency only, and therefore must always give way to efficacy. For example, if some gradient routine matches best the problem, it should be run, although sequentially.

All of the suggestions and conclusions made here have, possibly, been made elsewhere in the context of practical problems that were met and successfully solved, and therefore, considered one by one, do not contribute to the current scientific knowledge significantly. What is considered here to be the real added value of this dissertation is the presentation of those conclusions together. They are common to all the three presented practical cases.

Chapter 6

Summary; a proposal for automated simulation-optimisation problem solving

This chapter summarises all the observations reported so far in the preceding chapters. An important part of this summary is the recollection, in Section 6.1, of positive and negative evidence concerning simulation-optimisation problems solution that leads to formulation, in Section 6.2, of a proposal for an automated and universal environment suiting problems of the considered class. Justification for usefulness of such environment construction itself, as well as the argument for details of its workings, are to support Thesis 3. The dissertation closes with final remarks that are given in Section 6.3.

6.1 Motivation for automated simulation-optimisation problem approach

The proposal of an automated environment for simulation-optimisation problems solving is quite a natural consequence of facts presented so far. The results of analysis and attacking of the three practical problems, either satisfactory or disappointing, provide the main drive for such environment. They are reinforced by the fact that commercial modelling and optimisation tools, similar in their field of application, exist and thrive. Their presence should rather encourage and inspire creation of another optimisation environment being similar in idea but alternative in details. These two incentives are amplified by postulates formulated [19] by senior scientists and practitioners. Let us present the motives in detail.

Motivation coming from the three practical problems

Undoubtedly, the three practical simulation-optimisation problems considered here exhibit common features. The most distinct one is their performance index behaviour which (although this is visible at various magnification scale) demands the optimisation problem to be stated as a global one. The reasons for it may be the feasible region shape, or the performance index shape. Next, the optimisation task nature usually changes in the neighbourhood of an optimal solution: the initial obstacles vanish or, more commonly, get replaced by others like the simulation noise.

Changing problem features demand application of various optimisation routines. Fortunately, in cases when optimisation efficiency is not a prominent issue, only few methods suffice to serve most problems. Moreover, they do not have to incorporate advanced mathematics in order to carry out the task. EA, COMPLEX and SLP can be an exemplary triple that can be assumed to be in position to solve most of simulation-optimisation problems.

Simulation-optimisation problems are also united by a single universal solving approach, whose quintessence is to run routines currently best matching the problem in a sequence, and to make *ad hoc* routine modifications (alternatively, to employ a routine from outside the basic set) only under mounting difficulties. For the modifications to pay off those routines should remain simple to understand and inspect, simple and safe to change, able to be relinked with the rest of software.

The fact that simulation-optimisation problems share so many features clearly prompts for constructing a multi-purpose framework managing all problems similarly. The unresolved questions (e.g. the criteria for switching between routines) also encourage further research with purpose to determine how uniform can the switching rule be — and how much must be left to be defined by the user specifically for a considered problem.

Motivation by existence of commercial optimisation tools

Skillful merging of selected optimisation routines, performed by experienced practitioners, has already led to successful creation and marketing of tools for simulation-based optimisation. Products like *OptQuest*, *OptdesX* or *Epogy* have already been presented in Section 3.3, and *NOVA* has been mentioned on p. 85. They follow, although to various degree, the paradigm of switching between routines, of utilising all extra knowledge of the problem (if provided by the user), of allowing in-house simulation (and, sometimes, optimisation) code to be linked in. They are reported to be effective e.g. in design optimisation or in scheduling and management problems.

Therefore, it is quite natural that those products inspire the author to propose and prototype a simulation-optimisation environment of his own that hopefully would incorporate advantages and overcome disadvantages of the existing products. Among the advantages to be appreciated are procedures for setting parameters of optimisation routines (cf. *Epogy*). Also, gradient estimation under simulation noise is their valuable feature. On the other hand, there are things awaiting improvement: the support for simulation failures and user provision with detailed information what actually is going on. Generally, the disadvantage of of-the-shelf simulation-optimisation tools, as often expressed by author's fellow researchers, is their opaqueness and closeness that confines user activity only to using the product, which definitely does not suffice an academic.

Postulates for simulation-optimisation future as given by authorities

Experienced developers and researchers have been asked in [19] to express their position statements as to the future of simulation-optimisation given the current state of the art. The common opinion and postulate of the four authorities is that more effort should be put in making simulation and optimisation work together, and in not treating each other as a black box. Specifically, Royce O. Bowden distinguishes six domains of simulation-optimisation where the cooperation should take place:

1. Interfaces — addresses interfacing between simulator and optimisation routine, and between optimisation routine and human user.
2. Problem formulation — addresses construction of the performance index and constraints.
3. Methods — addresses the set of methods used conventionally in simulation-optimisation problems.
4. Classification — addresses the process of classification of methods that are suitable to a specific problem.
5. Strategy and tactics — addresses the most efficient use of available information and resources.
6. Intelligence — addresses the intelligence embedded in the solver to select the strategic approach and tactical employment of various techniques based on the problem classification.

The above postulates do not contradict the approach proposed on p. 82. In fact, both suggest the utilisation to the maximum extent of the available information about the model (Domains

3, 4 and 6) and the available parallel computation environment (Domain 5). Possibly, the key domain which enables formulation of the others, is developing the common simulation-optimisation interface (Domain 1). Existence of such interface conditions further automation of simulation-optimisation process as a universal routine, and not in the context of a concrete problem. However, such an ambitious task of simulation-to-optimisation and optimisation-to-user interfaces standardisation cannot be carried out without involvement of some influential consortium, still to appear.

6.2 Simulation-Optimisation Framework proposal

Simulation-Optimisation Framework (SOF) is the name of a computation environment proposed and prototyped by the author. It has emerged from the motivations given above. The following assumptions lie at its bottom:

- It is possible to distinguish a set of basic simulation-optimisation features; they are not many (like there are not many different routines required to solve a problem);
- It is possible to distinguish a set of basic optimisation routines that will be able to solve most simulation-optimisation problems.
- It is possible to come up with a common scheme for selecting the routine actually best fitting the problem.

Moreover, to make SOF opened to experiments and amendments, the following extra guidelines must be followed:

- SOF structure must allow to introduce easily user-delivered optimisation routines, and to override the built-in behaviour (e.g. to supply user-defined routine ranking algorithm).
- To provide environment elasticity, SOF core should be made without any particular assumptions about the nature of problems and about the optimisation routine specifics.
- Environment design must allow for execution in parallel and distributed systems.

Some of those postulates may actually seem contradictory (e.g. possibility to introduce non-standard algorithm switching strategy and the requirement that such switching should stay abstracted from the problem specifics). However, SOF proposal attempts to be a step forward from the current state of the art. Construction of such framework can have both didactic and practical outcomes. The didactic one is the necessity of distinguishing problem features in order to make use of the mechanisms for automated selection of an appropriate optimisation routine. The practical is hopefully a robust and open general-purpose optimisation engine.

Below are given the details of SOF design, starting from the outline, the implementation environment, through problem definition, to the switching routine. This description should rather be perceived as mental experiment since no practical results are available yet (although SOF prototype exists in its early version).

Selection of interface and development environment

SOF design must be performed carefully in order to follow all the above construction guidelines. Specification of interface to simulation and optimisation modules is the crucial point here. Definitely, all simulators should be interfaced uniformly in order to be uniformly managed by SOF core. The same applies to the optimisation module interface. On the other hand, those interfaces should be able to convey problem-specific information between simulation and optimisation — and this feature ought to remain transparent to SOF core.

Such interface specification as well as the selection of software technology used for implementation is not an easy task considered numerous limitations accompanying both solvers and simulators (cf. interface adaptation descriptions in Chapter 4 and in Chapter 5). Following the observation that it is the simulator that is usually the most opaque and inaccessible piece of software, minimum requirements must be imposed on its interface capabilities. It is the optimisation routine that must be charged with the costs of favouring the simulator — which is hopefully the correct decision since the suggested routines are destined to be modified (if needed) anyway, and they are publicly available in the source code form.

The selection of language for interface definition has as the main goals the easiness of coupling with the simulator and openness of the interface with purpose to supply upward compatibility. Java [1] has been dropped out despite its easiness and readiness for operation in distributed systems because its interface can be defined only statically. CORBA Interface Definition Language (IDL) has been selected as the language for interface specification. CORBA supports dynamic interface building and querying, which makes it possible for simulation and optimisation modules to define any extra methods. CORBA is simultaneously fully portable, unlike DCOM[2] — a competitive solution.

The core, the central module of SOF, as shown in Fig. 6.1, is the optimisation manager. This module task is to intelligently choose which optimisation routines from the candidate set will be currently run. Such defined manager role has the following implications to the inter-

[1]See Section B.2 for an overview of programming tools discussed here.

[2]Currently, the prototype version uses static interface and is implemented fully in C++.

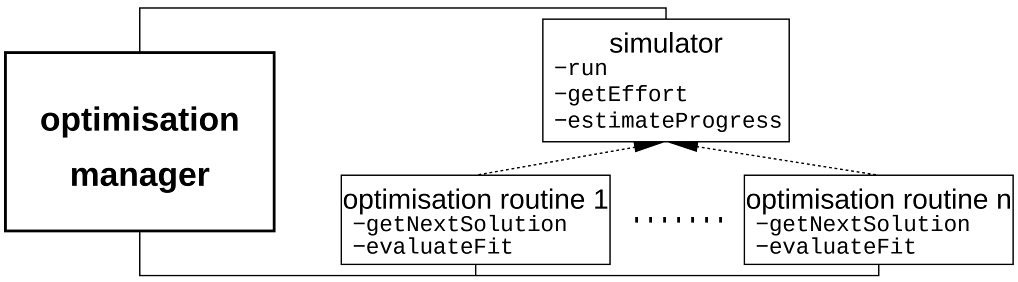


Figure 6.1: SOF structure. Optimisation manager maintains a single simulation module and a number of routines contained in optimisation modules. Method names are given using teletype font.

face definition. First, to provide a means of terminating one optimisation routine in favour of another one, the operations provided by an optimisation module should perform only one optimisation iteration. The consequence is that optimisation modules should keep their state between calls that are made in loop by the manager. Considered much better control over the optimisation code, such assumption does not seem excessive. (Otherwise, a preemptive optimisation would have to be assumed, which would have gravely affected the software design.) Second, in order to make it possible for the manager to perform initial ranking of the maintained routines (when no performance data are available yet) an optimisation module should provide a method assessing its own applicability to the problem. Third, this initial method-to-problem fit estimation should be verified and affected by the true method efficiency. It requires the simulator to provide methods calculating both progress and effort consumption made by the optimisation routine. Therefore, the interfaces to simulation and optimisation modules are proposed to be as follows:

The simulator:

`run` — Runs simulation for a given problem P.[3]

`getEffort` — Calculates computation effort of the last performed simulation.

`estimateProgress` — Evaluates quality of solution represented by problem P_{A} relative to another solution represented by P_{B}.

The optimisation routine:

`getNextSolution` — Runs one optimisation iteration and returns the new solution (represented by problem P_{k+1}).

[3]P is a set containing complete problem description: values of decision and dependent variables, definition of performance index, definition of constraints — and any other problem specific data. P appears to the optimisation manager as an amorphous piece of data. The manager main activity, i.e. best routine selection is based only on performance; the manager remains ignorant of the problem definition.

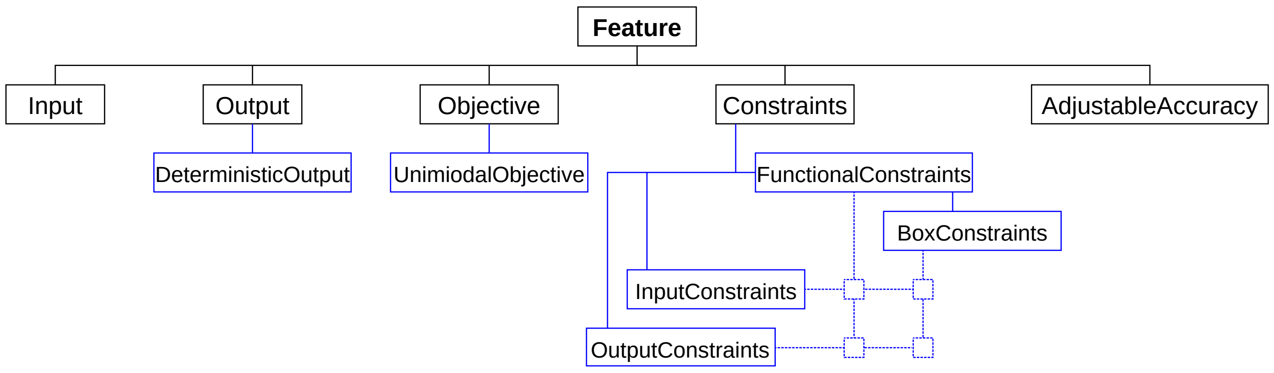


Figure 6.2: Hierarchy of SOF classes representing problem features that are considered to constitute the standard set. Further derivation of child classes specifying the problem description is possible (as presented in case of constraints by a mesh of unnamed squares).

`evaluateFit` — Evaluate its own fit to a problem P.

Note that such interface proposition conforms to the guidelines. Encapsulation of an existent simulator should not be a painful task: the only two extra requirements are to provide functions calculating simulation effort and progress (which, in the simplest case, can be based on simulation running time, resp. on the performance index value). Adapting an optimisation routine is more time taking: especially making it optimise iteration by iteration and evaluating quantitatively its fit to the problem may require some knowledge and some discipline in order to maintain assessing criteria the same for all solvers.

The fact that simulator is stateless and available through CORBA interface greatly facilitates parallel and distributed implementation. One of possible parallelisation techniques is to install a number of simulators in arbitrary locations, and to make them available for optimisation routines through a proxy, thus making a parallel virtual machine. Similarly, optimisation routines can be run anywhere, and in particular they may serve only as an interface to some local proprietary solvers.

Problem features classification

The key for SOF elasticity and openness should be the way a problem is defined. A set of basic types of simulation-optimisation problem attributes can be distinguished; each of those attributes can be represented as an object with well defined interface. Attributes form a hierarchy (implemented using multiple object inheritance), where a subclass usually defines a narrower problem than its superclass. The proposed set of basic simulation-optimisation attributes is presented in Fig. 6.2. All features inherit from abstract `Feature` class. Let us describe them shortly:

`Input` — Contains a vector of decision variables being the simulation input.

`Output` — Contains a vector of dependent variables being the simulation output. The output may be affected by a sort of randomness; to define a subclass of deterministic problems, specific `DeterministicOutput` subclass is proposed.

`Objective` — Pure abstract class; provides prototype of a method calculating the performance index value for a given problem P. Also, abstract `UnimodalObjective` subclass is provided, which should be used for problems known to have a unique optimum. User-provided classes should implement the performance index calculation methods.

`Constraints` — This abstract class and its subclasses provide means of specifying problem constraints, starting from the most general 'satisfied/violated' verdict, through constraints of various degree of regularity (`FunctionalConstraints`, `BoxConstraints`) applied for various kinds of variables (`InputConstraints`, `OutputConstraints`). This hierarchy can be extended; multiple inheritance can be used to construct classes defining precisely constraints for a given problem.[4]

`AdjustableAccuracy` — Contains a vector of simulation parameters, particularly those influencing the precision of calculations.

The above set of features suffices to distinguish between the basic classes of problems (stochastic/deterministic, local/global, constrained plainly/implicitly, with/without simulation accuracy control). This hierarchy can further be expanded and standardised, if needed. The standard features define all that simulation-optimisation problems may have in common. The only thing cannot be made a standard feature is the simulation itself, i.e. the definition of (1.2) or (1.3), which is the essence of the simulation-optimisation problems.

Therefore, optimisation works as follows. First, problem features are constructed, and packed into a set P. Next, the interplay follows Fig. 1.1: the currently selected optimisation routine changes `Input` contents and passes P to the simulator; the simulator changes `Output` contents and passed P back to the routine, where the performance index is estimated by `Objective` — and so on. One may ask about the point in dragging *all* problem information, contained in P, back and forth between the simulator and the optimisation routine. The explanation is that any optimisation routine is liable to change problem definition (e.g. to detect the optimum proximity — and to reformulate the problem as global and linearly constrained).

[4]The set of a class superclasses may be retrieved in the course of program execution using run-time type information (RTTI) mechanism, and used e.g. by `evaluateFit` method to infer about the problem specifics.

Since at any time the optimisation manager may decide to switch to another routine, the actual problem full definition must be contained within P.

Optimisation routines ranking and switching

There is a dilemma, which candidate algorithms should be given privilege to use the limited resource, the simulator? It becomes even more crucial if no more than one instance of simulator is available, and also because SOF is not a complete preemptive environment capable of suspending a routine invoking too many simulations in one optimisation iteration. For the answer, `evaluateFit` must be called at the very beginning for every candidate optimisation routine to make the first assessment of that routine applicability to the problem. What `evaluateFit` does is that it calculates a vector $\mathbf{z}$ of values $z_i \in \{-1\} \cup < 0, \infty)$; each z_i represents an estimate of how well the i-th feature can be handled (the bigger z_i the better). (The value of -1 means that a routine is completely unfit and should be discarded.) Two remarks should be made on this fit estimation procedure. The first is that such initial fit can really reflect a routine potential superiority due to e.g. utilisation of problem extra information the other routines are not aware of. Consider for example a case where `UnimodalObjective` in Fig. 6.2 is inherited by, say, `UnimodalObjectiveWithGradient`, a gradient information available from the simulator. Routines incapable of using that piece of information directly would customarily set the z_i corresponding for the base feature, `Objective`, to 1. Instead, a gradient-aware routine can set z_i to $O(\dim \mathbf{x})$, expecting performance superior to non gradient-aware methods. The second remark (somewhat contradictory to the former one) is that in the course of optimisation those initial fits get soon completely overridden by the actual efficiency of routines. Such distrust to initial fit estimates must exist in order to anneal potentially faked fits assumed by over-optimistic routine makers. Unfortunately, this fit estimation scheme is not very robust as it requires concerted action performed in all optimisation routines, which happen to be of various provenance. Hopefully, the following resource-to-routine allocation procedure could mitigate this problem.

The resource (i.e. the simulator) allocation to candidate optimisation routines is suggested to work as follows. Resource allocation is done in turns. In each turn k, every optimisation routine i that is in play is given computation credit $c_{k,i}$, thus making the overall i-th computation credit $\hat{c}_{k,i} = \hat{c}_{k-1,i} + c_{k,i}$. Next, `getNextSolution` is invoked in a loop, with the overall effort $\hat{e}_i$ being accumulated, until $\hat{e}_i \geq \hat{c}_{k,i}$ (i.e. until computation credit is exhausted). The question is what rules should the resource allocation algorithm follow. Naturally enough, it should promote routines that are best progress makers, or most efficient progress makers. The second option seems more appropriate as new algorithms come into play in the course of optimisation, since it is unfair to judge them after their absolute progress. Therefore, the allocation criterion based on the routine current efficiency should be applied. At the same time, potentially inefficient and initially badly ranked routines ought to be given some lump sum of computation budget in order to verify initial fit estimations practically. In SOF prototype, each routine is allocated a credit

$$c_{k,i} = \bar{e}_{k-1} \min\left(\frac{\eta_{k,i}}{\sum_{j=1}^{N} \eta_{k,j}} N,\ \ 1 \right) \quad , \tag{6.1}$$

where $\eta_{k,i}$ is i-th routine current efficiency (explained later), and $\bar{e}_k$ is the average effort per one `getNextSolution` call, calculated for all N currently used optimisation routines. Therefore, the allocation unit is the average effort per optimisation step, and the amount allocated is either 1 or a value proportional to the current efficiency ratio to the sum of all efficiencies.

Efficiency definition is a delicate issue. Recall conclusions from Chapter 4 and from Section 5.1 that the mechanisms used there did not work properly. The lesson learned is that milestones generated by so defined efficiency were unreliable and often unrelated to actual progress. The conclusion may be that efficiency should be 'more smooth' and possibly not based just on fixed width history window w. That is why fading effect is applied to efficiency computation in SOF. The efficiency of the i-th optimisation routine in k-th turn is defined recursively as

$$\eta_{k,i} = \frac{p_{k,i} + a\eta_{k-1,i}\tilde{e}_{k-1,i}}{e_{k,i} + a\tilde{e}_{k-1,i}} \quad , \qquad \text{with} \qquad \tilde{e}_{k,i} = e_{k,i} + a\tilde{e}_{k-1,i} \quad , \tag{6.2}$$

where $p_{k,i}$ is the progress made in one turn, $\tilde{e}_{k,i}$ is the discounted effort calculated for turn k, and a is the discounting ratio. Such efficiency definition causes big efficiencies experienced in the initial optimisation phase to be gradually forgotten, thus making it possible to compare routines that were in run for long with those freshly introduced. Obviously, this scheme has shortcomings: a routine that steadily makes small improvements is preferred over the one which, after ineffective exploration phase (like in EA), comes up with a solution far better than what the former routine has 'minced'. This phenomenon can be controlled by adjusting a, or by providing — through `estimateProgress` — an alternative definition of progress.

The suggested procedure for selecting routines to be affected by the simulator resource allocation is as follows. As each routine is attributed by a vector of fits, their Pareto subset can be found. Next, each of the Pareto methods is introduced into resource allocation scheme,[5] one routine in each turn. When all Pareto methods are already in play, another Pareto subset is found, after some delay, among the remaining methods. The procedure is repeated until all

[5]A routine is initialised with the problem definition taken from the routine that made the biggest progress (calculated by `estimateProgress`) computed w.r.t. the initial problem definition P_0.

available routines (except those branded with -1's) are in. Thus, at the end of the day, every routine declaring the faintest ability to solve the problem, is given its chance.

It was mentioned that a problem definition is likely to be changed, in the course of optimisation, by any optimisation routine. Such change would mean that the whole procedure of routines classification would start from scratch, or rather from the modified problem. This is a fragile spot of the whole algorithm since phenomenon of problem redefinition is given the highest precedence, and cancels so far collected initial fits and efficiency statistics. Therefore, it should be used in optimisation routines sparingly. Alternatively, one can consider a change of the managing algorithm so that problem change would be not so catastrophic, e.g. it may cause only a small part of simulation resources to be delegated for solving of such a changed problem.

Simulation-Optimisation Framework outline given here aims to support Thesis 3. This thesis claim of classifying problems by attributing them with properties coming from a standard set is met here by a proposition of problem standard features as in Fig. 6.2. Thesis claim for selection of core candidate optimisation routines can be satisfied in many ways if efficiency is not the major issue. Alternative proposals exist (see p. 80); the author's proposal is to use EA, COMPLEX and SLP routines.[6] Thesis claim for automated solver classification and switching is satisfied by SOF optimisation manager design; the proposed system architecture is portable, parallelisable, universal and expandable. SOF prototype tests have been made for simple problems and their results were promising.

6.3 Summary

With the growth of computing capabilities, the domain where heavy numerics is being applied is expanding, affecting branches more and more unfamiliar for optimisation practitioners. Therefore, when there appears the need to couple simulation with optimisation, the gap between the involved parties is very big: very often optimisation practitioners are not able to adopt all the knowledge of their colleagues, simulation practitioners, in order to develop an algorithm tailored to the specific problem. On the other hand, simulation practitioners, when trying to apply, on their own, an advanced optimisation procedure to their problem, are not trained enough to control all its intricacies. What they do then is to apply simpler optimi-

[6]As it was reported in Section 5.1, EA was once applied by IR to the waveguide design problem. The reason for IR's interest in CRS was EA's poor efficiency; it does not, however, means that EA is incapable of solving that problem altogether. Therefore, suggesting EA as the core optimisation routine does not contradict the results presented in Section 5.1.

sation routines, mostly heuristic, deprived of underlying complex mathematics. Usually they succeed because the models they use anyhow generate problems intractable by the advanced procedures.

The cases presented in this dissertation support the above observation. Most of the author's fellow researchers did not receive profound education in optimisation theory. Neither was it needed to solve their problems; after all, the algorithms that have been applied were conceptually much simpler and theoretically much less elaborate than classic advanced methods like GRG or SQP. Fortunately, it turns out that they, especially if run in parallel, may cope with the presented problems.

Probably, scientists in cases like those presented here will go for developing and adapting such simple optimisation routines by themselves, without consulting an optimisation professional. And they will be successful, especially with methods as robust as EA is. In such situations this dissertation attempts to suggest them an easiest possible approach to a simulation-optimisation problem. The dissertation also tries to make out of the three considered cases a common software structure, and to point out how to utilise such structure in creation of a universal simulation-optimisation framework. It is hoped that in the future SOF, probably not destined to become any standard, will encourage the colleagues from scientific community to discuss legitimacy of such standardised approach to simulation-optimisation problem. To author's knowing, no such open-source working environment exists, or at least none has been reported in the literature. On the other hand, there is much rather spoken evidence that simulation-optimisation problems are easily solved with multistage hybrid optimisation algorithms.

The interesting points of this dissertation, apart from the proposed approach itself, can be the details of COMPLEX procedure adaptation to handle simulation failures and the detailed studies of switching criteria for power system models optimisation. Out of probably a number of weak points of this dissertation, the author himself indicates notoriously poor theoretical background of the proposed optimisation algorithms and the resulting lack of support for the proposed approach other than through the three case studies. However, correctness of theses stated here can possibly be verified by the future developments in the field. Quite important contribution of this thesis is the demonstration that feasibility constraints — like those present in IHE simulator — may, if carefully handled, not prohibit the finding of an optimal (or, at least, good) feasible practical solution.

This summary is still to answer the anxious question about the fate of optimisation specialists, apparently not so much required for cases presented here. There are two options, both optimistic. The first is, that most *ad hoc* made simulation-optimisation software will some day become commercial products[7] and, leaving the stage of prototype, will have to be equipped with tailored optimisation routine. The second is that the existing solvers will expand their modelling capabilities and will be used as unified tools for simulation and optimisation. Both possibilities require involvement of high-class optimisation theoreticians and practitioners.

[7]Including any software made by the academic community.

Appendix A

Stochastic optimisation

Conventionally, the objective of stochastic optimisation is to minimise the expected value of performance index being the function of both decision and random variables

$$\min_{\mathbf{x}} \mathop{\mathbf{E}}_{\boldsymbol{\xi}} f(\mathbf{x}, \boldsymbol{\xi}) \quad , \tag{A.1}$$

where $\boldsymbol{\xi}$ is some random variable of finite variance. Such problem formulation dominates in the literature. Therefore, (A.1) is a stochastic analogue of (3.1). Virtually any deterministic method can be employed for stochastic optimisation provided that the function value and derivatives are replaced by their appropriate estimates [86, p. 22] — but such an ample switch from the original algorithm to its stochastic counterpart is usually inefficient. Therefore, numerous algorithms have been designed specifically for stochastic optimisation — and have become subject of many overviews [86, 3, 43].

As the result of literature searches, the view emerges that stochastic optimisation problems are presented and solved in two ways. The first group are cases where performance index is, in some way, regular and, consequently, great significance is placed by the authors on theoretical foundations of the applied optimisation routines. The second group is constituted mostly by discrete problems, with the emphasis put on successful combination of various optimisation, modelling and approximation techniques used for solution finding. In this appendix, optimisation algorithms are classified according to the type of information and assumptions they need to operate upon. An important category, called stochastic approximation (SA) methods, follows the well known Cauchy's steepest descent scheme that requires both objective function and gradient values to be available. Another approach, called response surface methodology (RSM), assumes that the performance index can be approximated by a deterministic function from a certain class. Those algorithms which operate with only the performance index value available, making no particular assumptions of the shape of the performance index, form the third category of the direct search methods. Except for SA, the optimisation routines present in those categories are either fundamental or have already been presented in Chapter 3, and therefore Section A.2 (RSM and similar techniques) and Section A.3 (direct search methods) will focus on necessary adaptations and successful application of those routines for optimisation problems. Instead, SA will be presented in full detail in (Section A.1): theory, modifications and applications.

A.1 Stochastic approximation

Stochastic approximation methods solve continuous stochastic simulation-optimisation problems as formulated in (A.1) by utilising a random estimate $\hat{\mathbf{g}}(\mathbf{x}, \boldsymbol{\xi})$ of the objective function gradient $\mathbf{g}(\mathbf{x}) = \nabla_{\mathbf{x}} \mathbf{E}_{\boldsymbol{\xi}}\, f(\mathbf{x}, \boldsymbol{\xi})$. Algorithms of this type generate a sequence $\{\mathbf{x}_n\}$ of problem solution estimates, with the generic formula for one algorithm iteration being

$$\mathbf{x}_{n+1} = \mathbf{x}_n - a_n \hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi}) \quad , \tag{A.2}$$

where $\{a_n\}$ is a sequence of positive step sizes such that

$$\sum_{n=1}^{\infty} a_n = \infty \quad \text{and} \quad \sum_{n=1}^{\infty} a_n^2 < \infty \quad . \tag{A.3}$$

The above algorithm should converge to a point where $\hat{\mathbf{g}}(\mathbf{x}) = \mathbf{0}$, which does not necessarily have to be the optimum $\mathbf{x}^\star$ of $\mathbf{E}_{\boldsymbol{\xi}}\, f(\mathbf{x}, \boldsymbol{\xi})$. However, this original SA scheme is rarely applied without further preconditions or improvements. To ensure that indeed the routine (A.2) converges to the global minimum additional assumptions are required, in particular that the performance index and the optimisation domain are convex. This, however, does not yet ensure the proper convergence if the performance index grows faster than quadratically when $\mathbf{x}$ changes. Moreover, such basic optimisation scheme still does not support the constraints on $\mathbf{x}$. Also, the proper choice of $\{a_n\}$ affects the algorithm convergence.

Gradient estimation techniques

There exist three basic gradient estimation procedures used by SA: finite differences, infinitesimal perturbation analysis (IPA) and likelihood ratio method (LR). For another one, the frequency domain experimentation, see [57].

Applying the universal method of finite differences, as requiring the least information of $f(\cdot)$, is often the only possible approach. However, this estimator has a big variance, is biased and computationally demanding. The estimate is obtained either by forward difference

$$\hat{\mathbf{g}}(\mathbf{x}, \boldsymbol{\xi}) = \sum_{i=1}^{\dim \mathbf{x}} \mathbf{e}_i \frac{f(\mathbf{x} + c\mathbf{e}_i, \boldsymbol{\xi}) - f(\mathbf{x}, \boldsymbol{\xi})}{c} \tag{A.4}$$

or by central difference formula

$$\hat{\mathbf{g}}(\mathbf{x},\boldsymbol{\xi}) = \sum_{i=1}^{\dim \mathbf{x}} \mathbf{e}_i \frac{f(\mathbf{x}+c\mathbf{e}_i,\boldsymbol{\xi}) - f(\mathbf{x}-c\mathbf{e}_i,\boldsymbol{\xi})}{2c} \quad . \tag{A.5}$$

Here, $\mathbf{e}_i$ denotes a versor along the i-th axis. Formulae (A.4) and (A.5) require $\dim \mathbf{x}+1$ and $2\dim \mathbf{x}$ evaluations of $f(\cdot)$, respectively. The optimisation scheme (A.2) with a finite differences formula applied for gradient estimation is known as Kiefer-Wolfowitz (KW) algorithm.

An important improvement to the above scheme was proposed in [107]. Instead of $\dim \mathbf{x}+1$ or $2\dim \mathbf{x}$ evaluations of $f(\cdot)$, it requires only $2m$ evaluations, m being considerably smaller than $\dim \mathbf{x}$. The idea is not to make an estimation of $\hat{\mathbf{g}}$ by separate shifts of $\mathbf{x}$ in each direction, but to perturb $\mathbf{x}$ simultaneously in all directions m times, and then to calculate the mean

$$\hat{\mathbf{g}}(\mathbf{x},\boldsymbol{\xi}) = \frac{1}{m}\sum_{i=1}^{m} \left(\frac{f(\mathbf{x}+c\boldsymbol{\delta}_i,\boldsymbol{\xi}) - f(\mathbf{x}-c\boldsymbol{\delta}_i,\boldsymbol{\xi})}{2c} \sum_{j=1}^{\dim \mathbf{x}} \frac{\mathbf{e}_j}{\mathbf{e}_j^T \boldsymbol{\delta}_i} \right) \quad . \tag{A.6}$$

In (A.6), $\boldsymbol{\delta}_i$ is a realisation of some zero-mean random variable, and c is a scaling factor (actually, c is not constant but predetermined for each algorithm step n, usually $c_n = c/n^\gamma$, $c,\gamma > 0$). By applying simultaneous perturbation gradient approximation method, significant speed-up of computations can be achieved, especially for moderate and large scale problems. An exemplary practical application of this routine is reported in [63].

IPA and LR methods do not require running dedicated simulations to estimate the performance gradient. Both procedures are based on the observation that the present results of, say, N simulations (i.e. N realisations of $\boldsymbol{\xi}$) can be used not only for computation of the performance value but for computation of the gradient as well. For the considered class of problems calculation of the performance value usually involves some kind of averaging, and the intermediate results (i.e. 'paths' of a discrete event dynamic system) are left unused by the finite differences method. In IPA and LR, by the careful inspection of those results one can infer about the behaviour of the system if $\mathbf{x}$ were going to change. The advantage of IPA and LR is that they often yield unbiased and consistent estimates but they require knowledge of the structure of the simulated stochastic system — namely the cumulative distribution function for the random variable $\boldsymbol{\xi}$ is supposed to be known.

Formally, both IPA and LR start off conceptually from the formula for the gradient of the performance function

$$\nabla \mathop{\mathbf{E}}_{\boldsymbol{\xi}} f(\mathbf{x},\boldsymbol{\xi}) = \nabla \int f(\mathbf{x},\boldsymbol{\xi}) p(\mathbf{x},\boldsymbol{\xi}) d\boldsymbol{\xi} = \int p(\mathbf{x},\boldsymbol{\xi}) \nabla_{\mathbf{x}} f(\mathbf{x},\boldsymbol{\xi}) d\boldsymbol{\xi} + \int f(\mathbf{x},\boldsymbol{\xi}) \nabla_{\mathbf{x}} p(\mathbf{x},\boldsymbol{\xi}) d\boldsymbol{\xi} \quad , \tag{A.7}$$

where $p(\cdot)$ is probability density function (p.d.f.) of the random variable $\boldsymbol{\xi}$. Since in general both $f(\cdot)$ and $p(\cdot)$ can depend on $\mathbf{x}$, the formula splits finally into the sum of two integrals.

The former integral can be estimated by computing the mean $\frac{1}{N}\sum_{i=1}^{N}\nabla_{\mathbf{x}}f(\mathbf{x},\boldsymbol{\xi}_i)$ for the same realisations $\boldsymbol{\xi}_i$ of $\boldsymbol{\xi}$ that were used for performance computation. The handling of the latter depends on the method: IPA transforms the problem to make it zero altogether, while LR computes it utilising still the same realisations of $\boldsymbol{\xi}$.

In LR (also known as score function method), the second integral is rewritten as follows

$$\int f(\mathbf{x},\boldsymbol{\xi})\nabla_{\mathbf{x}}p(\mathbf{x},\boldsymbol{\xi})d\boldsymbol{\xi} = \int f(\mathbf{x},\boldsymbol{\xi})\frac{p(\mathbf{x},\boldsymbol{\xi})}{p(\mathbf{x},\boldsymbol{\xi})}\nabla_{\mathbf{x}}p(\mathbf{x},\boldsymbol{\xi})d\boldsymbol{\xi} = \int p(\mathbf{x},\boldsymbol{\xi})\left(f(\mathbf{x},\boldsymbol{\xi})\frac{\nabla_{\mathbf{x}}p(\mathbf{x},\boldsymbol{\xi})}{p(\mathbf{x},\boldsymbol{\xi})}\right)d\boldsymbol{\xi} \tag{A.8}$$

so as to resemble the first integral in (A.7). Now it can be estimated by computing the mean $\frac{1}{N}\sum_{i=1}^{N}f(\mathbf{x},\boldsymbol{\xi}_i)\nabla_{\mathbf{x}}p(\mathbf{x},\boldsymbol{\xi}_i)/p(\mathbf{x},\boldsymbol{\xi}_i)$ for the same realisations of $\boldsymbol{\xi}$ as previously.

IPA represents $f(\mathbf{x},\boldsymbol{\xi})$ with $\boldsymbol{\xi}$ of any type (in particular, depending on $\mathbf{x}$) in an alternative way — by a function $F^{-1}(\mathbf{x},\boldsymbol{v})$ such that $\boldsymbol{v}$ is a random variable uniformly distributed in $< 0,1 >^{\dim\boldsymbol{v}}$ (and, certainly, independent of $\mathbf{x}$). Under these conditions, and following the rules from (A.7), we have

$$\nabla \underset{\boldsymbol{\xi}}{\mathbf{E}} f(\mathbf{x},\boldsymbol{\xi}) = \nabla \underset{\boldsymbol{v}}{\mathbf{E}} F^{-1}(\mathbf{x},\boldsymbol{v}) = \nabla\int F^{-1}(\mathbf{x},\boldsymbol{v})d\boldsymbol{v} = \int \nabla_{\mathbf{x}}F^{-1}(\mathbf{x},\boldsymbol{v})d\boldsymbol{v}\ . \tag{A.9}$$

The above equation appears simpler than formulae for LR (A.7, A.8) because all the analytical workload is hidden in the formulation of $F^{-1}(\cdot)$, which is possible only if one knows how to make $\boldsymbol{\xi}$ out of $\boldsymbol{v}$. It was observed that differentiating $F^{-1}(\cdot)$ resembles tracing system behaviour in presence of infinitesimally small changes of $\mathbf{x}$, and that is why IPA earned its name. The gradient in IPA is approximated by computing the mean $\frac{1}{N}\sum_{i=1}^{N}\nabla_{\mathbf{x}}F^{-1}(\mathbf{x},\boldsymbol{v}_i)$, where $\boldsymbol{v}_i$ are realisations of $\boldsymbol{v}$, analogously to $\boldsymbol{\xi}$.

LR is applicable to a larger class of problems than IPA but at the cost that estimates obtained from it tend to have larger variances. The optimisation scheme (A.2) with any unbiased gradient estimate applied is known as Robbins-Monro (RM) algorithm. A comprehensive overview of gradient estimation methods can be found in e.g. [6, p. 312 and on] or in [86, p. 231 and on].

Variance reduction

Certainly, keeping the variance of both objective and gradient estimates as small as possible improves the algorithm convergence. To attain this, the following variance reduction techniques are used: control variates, conditioning, stratified sampling, importance sampling and antithetic random variables. They all are based on the same idea of utilising some information about the model to reduce the variance of the estimator. Different techniques use different information to reduce the variance. Control variates technique redefines the estimator of

$\mathbf{E}_{\boldsymbol{\xi}} f(\mathbf{x}, \boldsymbol{\xi})$, involving in it some other random variable $\boldsymbol{\zeta}$, observable during simulation, whose expected value is known exactly. By appropriate coefficient setting one can get reduction of the variance, provided that $f(\cdot)$ and $\boldsymbol{\zeta}$ are related. Conditioning technique is applied when the procedure of computing the value of $f(\cdot)$ can be split logically in two phases: first, when some random variable $\boldsymbol{\varphi}$ can be observed, and second, when $f(\cdot)$ is generated from $\boldsymbol{\varphi}$ using some known conditional distribution. Instead of observing only the final result, one observes $\boldsymbol{\varphi}$ and directly calculates the conditional expectation. Stratified sampling technique is in some way complementary to conditioning: here the detailed distribution of the intermediate observation of $\boldsymbol{\varphi}$ is known, and the simulation needs to be started only to complete the calculation of $f(\cdot)$. In practice the simulation is run once for each of the values that $\boldsymbol{\varphi}$ can take and its results, along with the known distribution of $\boldsymbol{\varphi}$, serve to calculate the estimator of $\mathbf{E}_{\boldsymbol{\xi}} f(\mathbf{x}, \boldsymbol{\xi})$. Importance sampling technique is used in cases when events that are unlikely in the simulation process contribute significantly the value of $\mathbf{E}_{\boldsymbol{\xi}} f(\mathbf{x}, \boldsymbol{\xi})$, as happens e.g. in Monte Carlo integration of peaky functions. A new sampling density is chosen there that puts more weight to an area that affects the estimator of $\mathbf{E}_{\boldsymbol{\xi}} f(\mathbf{x}, \boldsymbol{\xi})$ strongly. Antithetic random variables technique uses, in subsequent simulation runs, random variables having the same distributions as $\boldsymbol{\xi}$, but being correlated in such way that the estimator variance be reduced. For example, let $f(\cdot)$ be a monotonic function of $\boldsymbol{\xi}^T = [\xi_1\ \xi_2\ \ldots]$, $\xi_i \sim \mathcal{U}(0,1)$ — then having a single realisation of $\boldsymbol{\xi}$ one may run two simulations: one with $[\xi_1\ \xi_2\ \ldots]^T$ and another with $[(1-\xi_1)\ (1-\xi_2)\ \ldots]^T$ that are negatively correlated, and hope that variance of such pairs of $f(\cdot)$ will be smaller. A broader description of variance reduction techniques can be found e.g. in [86, p. 221 and on].

Apart from the above general techniques of variance reduction, for the purpose of comparing different systems there is a widely used practice of utilising the same realisation of random variable $\boldsymbol{\xi}$ in all iterations of algorithm (A.2), known as common random numbers scheme, and also referred to as sample path optimisation. Having applied the common random numbers, the optimisation problem becomes, in fact, a deterministic one, and can be solved by a variety of methods. However, methods working in this fashion require the number of elements of $\boldsymbol{\xi}$ (or the number of $\boldsymbol{\xi}$ realisations) to grow as the current solution approximation nears the optimum $\mathbf{x}^\star$, in order to ensure accuracy. The proof that (under certain assumptions) the solution $\mathbf{x}^{\star,\dim\boldsymbol{\xi}}$ of the corresponding deterministic problem (based on a sample path of length $\dim\boldsymbol{\xi}$) tends to $\mathbf{x}^\star$ for growing $\dim\boldsymbol{\xi}$ is given in [97]. In most of simulation-optimisation problems, varying size of $\boldsymbol{\xi}$ can be handled, and has a reasonable interpretation. If, for example, $f(\cdot)$ is the output from Monte Carlo integration routine, then, in unidimensional case, $\dim\boldsymbol{\xi}$ can be given the interpretation of sample points number that was used for the integration. In one more example, where $f(\cdot)$ is the performance of a queuing system, $\dim \boldsymbol{\xi}$ is the number of interarrival times, i.e. the number of served customers.

A number of papers can be found dealing with sample path optimisation; for example, general convergence properties of sample path algorithms with constant $\dim \boldsymbol{\xi}$ are discussed in [97]. Instead, in [101] the plain stochastic approximation scheme is considered, but with a vector of common random numbers being periodically augmented with new random elements. This operation improves accuracy of estimates as the algorithm is getting closer to the solution $\mathbf{x}^\star$, but it restores randomness to the problem. See also [102] for a study on convergence of methods working in this fashion, especially in the case when $f(\cdot)$ cannot be estimated otherwise than by Monte Carlo simulation.

Adaptation of the step size and precision

The question arises how to choose the precision of simulation (e.g. number N of points for Monte Carlo simulation) and the sequence $\{a_n\}$ of step sizes to maintain reasonable compromise between the solution accuracy and the computing effort. The original approach is to use a fixed simulation precision, and to define $\{a_n\}$ a priori

$$a_n = \frac{a}{n} \quad , \tag{A.10}$$

where a is some initial step size. It is useful to consider another formula

$$a_n = \frac{b}{c+n} \quad \text{where} \quad c \gg b > 0 \quad , \tag{A.11}$$

which prevents a_n from decreasing rapidly for small n. For (A.10), it is shown [86, p. 290] that if we consider a more general formula, $a_n = a/n^\alpha$, then $\alpha = 1$ will still be the best choice.

The same author proposes an adaptive procedure for choosing $\{a_n\}$ based on the observation that the step size is optimal if the gradients in two consecutive iterations stay orthogonal, i.e. $\hat{\mathbf{g}}^T(\mathbf{x}_n, \boldsymbol{\xi})\hat{\mathbf{g}}(\mathbf{x}_{n+1}, \boldsymbol{\xi}) = 0$. The proposed routine reduces a_n by half if the above dot product is going to be negative. Of course, for the formula to make sense, the gradient estimates have to be based on the same realisation of $\boldsymbol{\xi}$, at least for the purpose of comparing $\hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi})$ and $\hat{\mathbf{g}}(\mathbf{x}_{n+1}, \boldsymbol{\xi})$.

The postulate for adaptation of a_n can be found in many papers [119, 102, 122]. The general suggestion for a_n to be the result of exact line minimisation along $\hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi})$ in practical implementations is usually replaced by the more liberal Armijo rule that in the case of algorithm (A.2) takes form (iteration number mark n at $\mathbf{x}$'s and a's is omitted here for clarity)

$$\left|\hat{\mathbf{g}}^T(\mathbf{x} - a\hat{\mathbf{g}}(\mathbf{x}, \boldsymbol{\xi}), \boldsymbol{\xi}) \cdot -\hat{\mathbf{g}}(\mathbf{x}, \boldsymbol{\xi})\right| \leq -\epsilon \hat{\mathbf{g}}^T(\mathbf{x}, \boldsymbol{\xi}) \cdot -\hat{\mathbf{g}}(\mathbf{x}, \boldsymbol{\xi}) \quad . \tag{A.12}$$

This means that the product of the minimisation direction $-\hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi})$ and the gradient estimate at $\mathbf{x}_{n+1}$ does not have to be zero (as in exact minimisation case) but a small (delimited by ϵ) part of an analogous product at $\mathbf{x}_n$.

The authors cited above combine adaptation of a step size with the increasing simulation precision. Wardi and Shapiro [119, 102] assume that the size of the random variable (i.e. the number of random points passed to the simulator) grows to the infinity in a predetermined way as the algorithm proceeds, and show the convergence in such general case. Next, Yan and Mukai [122] propose the measures to monitor the progress of performance optimisation and of the estimation error, and use them to determine when the size of random variable $\boldsymbol{\xi}$ should be increased. The progress measure is based on the absolute difference $|f(\mathbf{x}_n, \boldsymbol{\xi}) - f(\mathbf{x}_{n+1}, \boldsymbol{\xi})|$ of the objective at two consecutive solution estimates for the same realisation of the random variable, and the error measure is based on the absolute difference $|f(\mathbf{x}_n, \boldsymbol{\xi}) - f(\mathbf{x}_n, \boldsymbol{\xi}^{+})|$ of objective at the same solution estimate and for the same realisation of $\boldsymbol{\xi}$, with $\boldsymbol{\xi}^{+} = \begin{bmatrix} \boldsymbol{\xi} \\ \tilde{\boldsymbol{\xi}} \end{bmatrix}$, where $\tilde{\boldsymbol{\xi}}$ represents new random elements generated in order to improve accuracy of $f(\mathbf{x}_n, \boldsymbol{\xi}^{+})$.

The simulation accuracy and the underlying dimension of random variable used to estimate the objective and its gradient depend on the kind of simulation. In this section mostly Monte Carlo simulations were considered. However, in [27] a case is discussed of optimising the service time in a single server queue with generally independent customer interarrival time (GI/G/1). It is proved that the next solution estimate $\mathbf{x}_{n+1}$ can be computed as soon as a new customer arrives, and the algorithm still converges.

The ideas of step size and simulation accuracy adaptation are particularly important when an approximation of the objective function is made that is more complex than linear. Methods based on such approximations are discussed on p. 150 in section 'Other improvements'.

Averaging

A very simple yet powerful modification to the classic SA algorithm that dramatically improves its convergence, has been invented simultaneously by Polyak and Ruppert. It is supported by appropriate proofs in [88], and one of numerical examples for its superiority can be found in [123]. The idea is that, instead of observing a sequence $\{\mathbf{x}_n\}$ of solution estimates, one can observe a sequence $\{\overline{\mathbf{x}}_n\}$ of their averages

$$\overline{\mathbf{x}}_n = \frac{1}{n} \sum_{i=0}^{n} \mathbf{x}_i \quad . \tag{A.13}$$

In this case, however, to preserve the algorithm's convergence, one has to replace the formula (A.10) or (A.11) with

$$a_n = \frac{a}{n^\alpha}, \quad 0 < \alpha < 1 \ . \tag{A.14}$$

The idea of averaging has been developed further in [65]. The authors propose that the averaging does not have to be performed over the whole history. They prefer to calculate a moving average

$$\overline{\mathbf{x}}_n = \frac{1}{w_n + 1} \sum_{i=n-w_n}^{n} \mathbf{x}_i \tag{A.15}$$

and to set the averaging window size w_n so that the convergence is preserved. Generally, w depends on α in (A.14), i.e. w decreases as $\alpha \to 0$ because more frequent oscillations of $\{\mathbf{x}_n\}$ around $\mathbf{x}^\star$ in the final phase of optimisation need less averaging.

Yet another interesting adaptation of the idea of averaging is presented in [31]. It is proposed to apply the basic averaging scheme (A.13) with the sequence of step sizes defined *a priori* but indexed in other way than just with an index of the current iteration. The indexing variable there is the counter of 'flips' of $\hat{\mathbf{g}}(\cdot)$ that happen nearby $\mathbf{x}^\star$. Therefore, the algorithm reduces its step size only in the final stage, when a vicinity of the solution has been reached.

Constraints

SA in its original version is definitely not suited for constrained optimisation problems — strong assumptions are made in the literature with respect to both the performance index and the optimisation domain. To ensure global convergence, performance index is required to be strictly convex, and domain is required to be closed and convex. Usually, it is assumed that domain is a cartesian product of the intervals of feasible values for each x_i, i.e. as it was defined in (1.5a). The common approach to handle constraints is to project the current estimate of $\mathbf{x}^\star$ on $D_\mathbf{x}$ in each routine iteration. Therefore, (A.2) gets replaced by

$$\mathbf{x}_{n+1} = \pi_{n+1}\left(\mathbf{x}_n - a_n \hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi})\right) \quad , \tag{A.16}$$

where $\pi_{n+1}(\cdot)$ is a constraint preserving operator.

Usually, $\pi_{n+1}(\tilde{\mathbf{x}})$ accepts $\tilde{\mathbf{x}}$ if $\tilde{\mathbf{x}} \in D_\mathbf{x}$, and preserves $\mathbf{x}_n$ otherwise (an example of such algorithm is given in [27]). For another type of projection, recall [119] — there, the domain is defined through a set of functions as in (3.2a), and projection is integrated into the line minimisation procedure: the step size a_n must be chosen so that none $h_{\mathrm{I},i}(\mathbf{x}_n - a_n \hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi}))$ becomes positive.

There is another, mentioned earlier, reason, for which the basic stochastic approximation scheme is so widely supplemented with the projection feature. It is because SA (A.2) does not converge if $f(\cdot)$ grows superlinearly. For badly chosen a and the starting point $\mathbf{x}_0$, the rate at which $\{a_n\}$ decreases is simply insufficient to overcome the growth of $\mathbf{g}(\cdot)$ and $\{\mathbf{x}_n\}$ starts to oscillate around the solution. It is not so rare a case: Fu [44] considers one-dimensional parameter optimisation problem for a GI/G/1 queue with the projection operator

$$\pi(x) = \begin{cases} x_{\mathrm{L}} & \text{if } x < x_{\mathrm{L}} \\ x_{\mathrm{U}} & \text{if } x > x_{\mathrm{U}} \\ x & \text{otherwise} \end{cases} , \tag{A.17}$$

where x_{L} and x_{U} are the bounds. Unfortunately, the projection operator (A.17) does not suffice because in the considered case $f(x) \to \infty$ as $x \to x_{\mathrm{L}}$ or $x \to x_{\mathrm{U}}$. It is then proposed to solve this difficulty by optimising over a region that is smaller than $< x_{\mathrm{L}}, x_{\mathrm{U}} >$. Such workaround causes another problem: how to reduce the search domain so that $x^\star$ still remains inside it?

A solution of this problem would be to change the size of the region on which $\mathbf{x}$ is projected in adaptive way. The proposal of such an adaptive projection was presented in [26]. First, a sequence $\{M_n\}$ of increasing radii is constructed, and a counter σ that stores the number of iterations in which the values $\mathbf{x}_n - a_n\hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi})$ fall off the circle $C(\mathbf{0}, M_\sigma)$. In case when the violation of this circular constraint would happen, $\mathbf{x}_{n+1}$ is reset to some predefined point $\mathbf{x}^+$. It means that the algorithm starts over from $\mathbf{x}^+$, but with a_n smaller and M_σ bigger than before, as the values of n and σ persist over the algorithm restarts. By choosing $\{a_n\}$ and $\{M_n\}$ skillfully, good convergence can be obtained. The idea presented in [26] has been further developed in [4]: the author improves the algorithm behaviour in the case when $\mathbf{x}_{n+1}$ is reset to $\mathbf{x}^+$. He proposes, instead of returning to $\mathbf{x}^+$, to project $\mathbf{x}_n - a_n\hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi})$ onto a predefined set. Proceeding in this way, not all progress made in the previous iterations is lost.

Other improvements

Since 1951, the classic SA algorithm has been modified and improved by many authors in other ways than discussed in the paragraphs above. Perhaps the most often adopted modification was to assume the step size a_n not to be a scalar but rather a matrix that would affect not only the step size but also the direction — like in the well known Newton's method:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - (\nabla^2 f(\mathbf{x}_n))^{-1}(\mathbf{x}_n)\nabla f(\mathbf{x}_n) \ , \tag{A.18}$$

where $\nabla^2 f(\mathbf{x}_n)$ is the Hessian matrix of $f(\cdot)$ at $\mathbf{x}_n$. Such improvement was proposed in [99, 120], and claimed to accelerate algorithm convergence. Actually, the authors have assumed that $\nabla f(\mathbf{x}_n)$ was observable directly, and formulated the optimisation problem as finding $\mathbf{x}^\star$ such that $\nabla f(\mathbf{x}^\star) = \mathbf{0}$. Their formula for $\{\mathbf{x}_n\}$ merges (A.2), (A.10) and (A.18) as follows:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \frac{a}{n}\hat{\mathbf{J}}^{-1}(\mathbf{x}_n, \boldsymbol{\xi})\hat{\mathbf{g}}(\mathbf{x}_n, \boldsymbol{\xi}) \quad , \tag{A.19}$$

where $\hat{\mathbf{J}}^{-1}(\mathbf{x}_n, \boldsymbol{\xi})$ is an estimate of $\nabla^2 f(\mathbf{x}_n)$ calculated using finite differences scheme. The latter paper [120] proposes two more additions to (A.19). First, the construction of $\hat{\mathbf{J}}^{-1}(\cdot)$ is based also on the values of $\hat{\mathbf{J}}^{-1}(\cdot)$ in the previous steps. Second, the identity matrix is used in steps where the Hessian matrix would be singular.

The paper [87], mentioned already, proposes an entirely different algorithm in which the quadratic problem (cf. p. 67) is solved in each step to find $\mathbf{x}_{n+1}$. That quadratic problem is deprived of any randomness, because all the necessary computations in a single step are performed for the same realisation of $\boldsymbol{\xi}$. The proposed approach is to find the subgradients $\boldsymbol{\varrho}_1, \boldsymbol{\varrho}_2, \ldots$ of $f(\cdot)$ at $\mathbf{x}_n$, thus constructing a cutting-plane approximation of the performance function. The problem passed to a QP solver is to minimise this approximation, which is additionally regularised so that $\mathbf{x}_{n+1}$ lies not too far from $\mathbf{x}_n$. Therefore the routine is as follows:

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \mathbf{d} \qquad \mathbf{d}, v = \arg\min\left(v + \frac{||\mathbf{d}_n||^2}{c_n},\ \ v \geq \boldsymbol{\varrho}_i^T\mathbf{d}\right) \quad , \tag{A.20}$$

where c_n is the regularisation coefficient. QP also allows to support any other explicit equality or inequality constraint that has been imposed on the original problem externally. Application of subdifferential broadens the suite of problems supportable by the algorithm.

A significant improvement of optimisation efficiency can be achieved not only through the algorithm amendments, but also by simplifying the simulator itself. Let us return once again to [87] for an example. The authors consider discrete tandem production lines, i.e. those producing separate workpieces. However, they use simulation routines suitable for the continuous products like fluids. By doing so, they reduce the simulation time by an order of magnitude at the cost of 4% error in $f(\cdot)$ relative to the discrete simulation case.

While certain researchers' goal is to reduce the variance, others deliberately make the optimisation routines wander for some time in $D_{\mathbf{x}}$ in search for the global, not local, solution. Such an algorithm is presented in [46]; there every new solution estimate is perturbed with multidimensional independent Gaussian random variable. The variance of this variable diminishes as the algorithm proceeds, making it less and less able to leave the attraction area of a local minimum. This algorithm resembles the simulated annealing scheme described on p. 73.

A.2 Response surface methodology and other approximation techniques

This section presents collectively selected techniques for performance index approximation. Although presented in context of stochastic optimisation, performance index approximation may serve two purposes. The first one is to get rid of indeterminism altogether by replacing the original performance index in optimisation problem with its deterministic approximation, constructed through linear regression procedure. The second one, arguably not supposed to be mentioned in here since it concerns deterministic optimisation, is to mitigate the burden of costly model solving by using a metamodel to approximate performance index from relatively few genuine model results.

Response surface methodology is the approximation technique used in stochastic optimisation to attain the first purpose. Unlike in SA, classic RSM does not require perpetual interaction between the optimisation routine and the simulator. The optimisation is performed in three steps:

STEP 1: Simulation is invoked at for a number of points from the domain. Those points can be chosen at random or using some predefined scheme. The results of simulations are passed to STEP 2.

STEP 2: The unknown function $\mathbf{E}_{\boldsymbol{\xi}}\, f(\mathbf{x}, \boldsymbol{\xi})$ is approximated by some known deterministic function $\tilde{f}(\mathbf{x})$, fit by regression, using the values calculated in STEP 1.

STEP 3: The minimum of $\tilde{f}(\cdot)$ is sought by some deterministic optimisation routine.

In most practical applications, steps 1–3 are executed cyclically. In subsequent executions, as the solution estimate approaches $\mathbf{x}^{\star}$, functions $\tilde{f}(\cdot)$ of more complex structure are being fitted (e.g. first linear, then quadratic, finally cubic). Also, the region from which the samples of $f(\cdot)$ are drawn, is being made smaller and smaller. According to [24], the response surface methods perform in general better than their stochastic approximation alternatives. In order to be effective, RSM — similarly to SA — requires considerable knowledge of the nature of the problem being solved. In SA, it was the gradient estimation that required deep insight into the model; here what comes from extra knowledge is the proper type of response surface approximation function.

RSM requires relatively few samples to construct accurate approximations of performance functions provided that those functions themselves are not very complicated. For rapidly changing performance RSM, regardless of the number of samples it is based on, does not suffice. This is quite a frequent case when deterministic optimisation problems are concerned. In such situations more elaborate approximation techniques are employed, capable of predicting behaviour of a quickly varying performance index in between sample points. Let us present three of them: artificial neural networks, Bayesian regression and Krigging.

Artificial neural networks, i.e. mathematical models inspired by structure and workings of brain, have been applied in numerous domains, and often over-optimistically. Their success in response surface approximation is, however, well-established. Once properly designed (w.r.t. the structure and the number of neurons), a neural network is able to achieve good compromise between prediction accuracy and generalisation properties — the features that remain somewhat contradictory. Potential pitfalls of neural networks are the phenomena of underfitting and overfitting, related to the number of neurons in the hidden layer, and consisting in insufficient or too literal utilisation of the training data by the network.

In Bayesian regression technique, the performance function is approximated at each point by random variable with Gaussian distribution rather than by a deterministic value. Such, seemingly pointless, introduction of indeterminism to the originally deterministic problem is a precious improvement since it gives information not only about the function value at point $\tilde{\mathbf{x}}$ but also some measure of approximation credibility there. Therefore, the introduced indeterminism stands for the uncertainty about performance index behaviour in areas between spots where samples were taken, and not for any inherent randomness present in the modelled system [30]. The distributions provided by Bayesian regression technique are the result of some statistics performed over all samples (e.g. on covariance calculations). The potential problem with Bayesian regression technique, as signalled by the literature, is excessive numerics, now overcome by massive fine-grained parallel computing.

Krigging (named probably after a Dutch mining geologist) is an interpolation technique locating itself in mid-way between RSM and Bayesian regression. Similarly to the latter one, local statistics are collected from the samples; namely, autocovariance is calculated. Next, the interpolation is performed by a moving weighted average process, where the weights are adjusted using the autocovariance so that expected squared estimation error is kept at its minimum. Krigging, initially developed for 2-D and 3-D spaces, has been extended to multidimensional version, and there is evidence of its proper working. However, for practical reasons, some standard formula is picked to describe the autocovariance instead of keeping it in its original form. Consequently, the fine theoretic properties of the method (i.e. the minimum estimation error) get spoiled.

A.3 Direct search methods

As it was already mentioned, every deterministic optimisation routine can be applied to solve stochastic optimisation problem provided that appropriately accurate estimate of $f(\cdot)$ is available. The direct search algorithms presented in this section are, in major part, adaptations of well-known algorithms presented through Section 3.2. The common problem in their adaptation is when actually resampling of $f(\cdot)$ must take place to ensure the convergence that is usually severely handicapped by fluctuations of $f(\cdot)$.

Simplex search

The ancestor of Nelder-Mead direct search routine (cf. p. 71), the classic downhill simplex search was probably the first algorithm adapted for stochastic optimisation problems. The adaptation was described in [108]; the routine used by the authors was still devoid of expansion and contraction operations — the simplex moved across the domain being transformed only by subsequent reflections. When the proximity of $\mathbf{x}^\star$ was reached then the simplex started to rotate around the solution, which was the sign to stop the procedure. Without any improvements that original simplex algorithm would probably get anchored far from $\mathbf{x}^\star$ by an occasional unusually good simulation result. To avoid that, the authors propose to repeat simulations — but only for 'stale' simplex vertices — with purpose to force the algorithm forward.

Similarly for the regular Nelder-Mead procedure: when it is employed with no improvements for stochastic problems, it terminates prematurely due to far too frequent contractions it happens to make. This drawback is addressed in [15]. The authors, after a brief review of the former algorithm amendments, point out the cases where resampling of $f(\cdot)$ solves the problem. They additionally advise making the contraction operation less strong. Yet another revised simplex procedure is proposed in [55], and numerical examples are provided showing its superior performance relative to the algorithm presented in [15]. The postulated changes include geometrical decrease of contraction coefficients and linear growth of expansion coefficients, as optimisation proceeds. Also, procedure restarts with preserving the actual best solution, are postulated.

Simplex search routines, like all routines where the selection of a trial point is driven by ranking of points in the pool, work well if the fluctuations of $f(\cdot)$ due to random factors are so small that they do not affect the ranking of the pool points. It is usually the case in the preliminary stage of optimisation. However, when approaching $\mathbf{x}^\star$, the ordering of points in X_n becomes distorted by the randomness of $\boldsymbol{\xi}$, and algorithms start oscillating around the solution. More accurate estimates of the performance index are needed at that point in order to make the optimisation progress any further.

Simulated annealing

Adaptations of simulated annealing algorithm (cf. p. 73) to stochastic problems use an estimate of the objective function value (mostly it is the mean of an increasing number of observations of $f(\cdot)$ at some trial point $\tilde{\mathbf{x}}$). Moreover, they are *greedy*, i.e. they preserve the best solution estimate found in the course of the algorithm run. Convergence properties of so adapted algorithms are discussed by Gelfand and Mitter [45]. Next, Fox and Heine [42] postulate some amendments that prevent simulated annealing from staying too long without a move, thus improving its efficiency.

Adaptation of the number of observations used to estimate the objective is proposed in [23]. It is postulated that the number of observations must be increased only if the difference $f(\tilde{\mathbf{x}}_{n+1}, \boldsymbol{\xi}) - f(\mathbf{x}_n, \boldsymbol{\xi})$ is small w.r.t. the current confidence intervals for $f(\tilde{\mathbf{x}}_{n+1}, \boldsymbol{\xi})$ and for $f(\mathbf{x}_n, \boldsymbol{\xi})$. Moreover, if the replication of observations of $f(\cdot)$ does not allow to make reliable comparison, then the temperature θ is reduced and another trial point is chosen. Such algorithm has been successfully applied for an exemplary problem of buffer sizes optimisation in transport systems. A similar algorithm is also described in [14], and applied for steelworks design parameters (finding optimal number of cranes, furnaces etc.) problem. In both cases the search domain is discrete, and a proper neighbourhood has to be constructed so that the algorithm is able to penetrate the whole search space.

Generally speaking, simulated annealing can be used in stochastic optimisation, both continuous and discrete, provided that the variance of the estimator of the objective function decreases to zero for growing number of observations. Also, proper definition of the neighbourhood is an important issue, sometimes requiring as much knowledge of the system as when creating appropriate distributions for gradient estimation in SA, or as when determining the response surface type.

Other algorithms

An algorithm quite similar to simulated annealing and frequently cited has been presented in [121]. Instead of dealing with the randomness of the objective function by averaging over an increasing number of samples, the authors propose to utilise the random character of the objective for the purpose of searching for the global optimum. In fact, the needed modifications of the basic simulated annealing scheme are few: the main is that instead of looking for the minimum of $\mathbf{E}_{\boldsymbol{\xi}}\, f(\mathbf{x}, \boldsymbol{\xi})$ one looks for such $\mathbf{x}$ that maximises the probability of $f(\mathbf{x}, \boldsymbol{\xi}) \leq \vartheta$. Here, ϑ is a random variable uniformly distributed over the interval $< f_{\mathrm{L}}, f_{\mathrm{U}} >$ of approximate values that $f(\cdot)$ can take. In practical realisation of this algorithm the value $f(\tilde{\mathbf{x}}_{n+1}, \boldsymbol{\xi})$ is measured M_{n+1} times at trial point against a sort of a 'stochastic ruler', i.e. against ϑ. If in all measurements the test $f(\tilde{\mathbf{x}}_{n+1}, \boldsymbol{\xi}) \leq \vartheta$ is passed, then the candidate $\tilde{\mathbf{x}}$ is accepted. Of course, for M_n growing with the progress of optimisation, the tendency to accept worse candidate points decreases, analogously to the classic simulated annealing scheme. The proposed algorithm converges under relatively mild conditions, and is simple to implement.

Another, still more general and more global algorithm was proposed in [5]. This routine is also based on the simulated annealing scheme (or rather, on the random walk scheme). It operates on the domain being a set of a finite number of points; for each of them the probability of being chosen as the next trial point $\tilde{\mathbf{x}}$ is equal. Therefore, no cooling scheme exists, but for each point from domain the number of times it was visited, is recorded. As the algorithm proceeds, the statistics are collected, and the point visited most often is considered to be the solution. In this routine the candidate $\tilde{\mathbf{x}}$ is allowed to become $\mathbf{x}_{n+1}$ only if $f(\tilde{\mathbf{x}}, \boldsymbol{\xi}) < f(\mathbf{x}_n, \boldsymbol{\xi})$. The convergence of this algorithm to the global optimum is demonstrated.

Quite often the engineers and researchers, inspired by widely recognised classic algorithms, develop their own routines, often tailored to the specifics of a particular problem. Those routines can combine the techniques existent since long time in separate algorithms into one piece of software. An example of such approach can be found in [110], where the problem of optimal network design is addressed. The author employs common random numbers scheme (to reduce the estimator variance), adaptation of sample size (to increase accuracy), perturbation analysis and, finally, the tabu search (cf. p. 75) paradigm, obtaining a method that performs very well, at least for the considered class of problems.

There are many other well known and widely applied routines for stochastic optimisation problems: tabu search, evolutionary strategies (cf. p. 76), importance sampling methods. For a short discussion on them and for further references, see [3, 24, 109].

Appendix B

Parallelisation tools and techniques

The postulate contained in Thesis 2d to speed up lengthy by their nature calculations seems quite trivial. However, one should consider here what actually can be subject to parallelisation and how this parallelisation can be effectuated for simulation-optimisation problems. Provided the two-module system structure, as in Fig. 1.1, computation speed-up can be achieved by parallelisation of simulation process and/or by parallelisation of optimisation process. Which operations should be parallelised depends on the nature of simulation and on the type of computing environment available. It must be emphasised that if massive parallel architecture computing power (represented by e.g. Single Instruction Multiple Data — SIMD) is at hand, it ought to be definitely used by the simulator — if only the simulation nature corresponds to that architecture (e.g. in FEM). Otherwise that architecture advantages, if left over for the optimisation routine, will bring only a marginal performance improvement, considered the mere ratio of effort needed for simulation and for optimisation. (They may be eventually used for matrix operations and alike, but the computing effort made by the optimisation routine still remains a minor part of the simulation effort.) If the simulation nature is so that such fine-grained parallelisation is not possible, the allocation of parallelisation techniques to simulator or to optimiser is more flexible.

The above suggestions can be opposed by the argument that the simulator is often a closed piece of software, and no intervention inside it is possible — especially affecting the way parallelism is used. This is true in many cases — but equally frequently the simulator is configurable to use SIMD architecture, or available with its source code, and it is its intricacies that scare off and make it 'closed'. Sometimes it is enough to recompile the simulator to make use of the architecture; there exist already compilers [56] capable of automated loop parallelisation with SIMD instructions. Alternatively, *OpenMP* [79, 117] mechanism can be used for coarser-grain parallelisation. *OpenMP* offers a set of parallelisation directives that may be inserted into an existing source code in spots that can be easily investigated with a software profiler. However, *OpenMP* spawns multiple threads for parallel execution instead of using SIMD instructions.

In cases when no parallelisation inside the simulator is possible, one has to consider the simulation as an atomic operation. Consequently, the only parallelisation one can think about is the coarse-grain optimisation routine parallelisation. Such parallelisation, unlike the fine-grain one, cannot be carried out automatically. Moreover, provided diversity of optimisation routines there is no single parallelisation methodology — each routine requires an approach of its own. Selected optimisation parallelisation approaches are presented below in Section B.1.

Third-party commercial simulators, irrespectively of their SIMD support, not only make the interference into the source code impossible but, moreover, introduce mechanisms (e.g. hardware keys, fixed directory structure, fixed communication port numbers) allowing only a single simulator instance to be run on a machine. This, apart from non-trivial coarse-grain optimisation parallelisation, leads to the need of distributing simulation over a number of machines. Techniques of such distribution are presented in Section B.2

B.1 Parallelisation of optimisation routines

According to the software engineer's involvement, three levels of routine parallelisation can be distinguished [62]:

Code level — simple loop spreading or data spreading in the original routine, performed either manually or through some automation tool;

Method level — more profound original routine modifications that take into account specifics of the parallel environment available;

Problem level — complete reconsideration of the original routine leading to profound changes or routine redesign, with execution in parallel as intrinsic feature of so constructed routine.

Some of techniques applied at code level have been already mentioned while speaking of simulation parallelisation (*OpenMP*, SIMD compilers). Some other, working in similar fashion, Fortran tools are worth to be recommended here, especially that much of existent optimisation software is written in that language. Those tools are Parallel Support Library (PSL), a thread library [117], and High Performance Fortran (HPF), a tool for targeting SIMD architectures by using parallelisation directives [53, 118]. As regards C and Java languages, code-level parallelisation is commonly achieved through spawning a multitude of threads.

Code level parallelisation, whenever available, is commonly used in optimisation routines for matrix operations or similar tasks, and therefore can be placed inside all gradient methods. However, as it was mentioned, it does not lead to any significant speed-up in optimisation until it is performance index evaluation calls that get invoked simultaneously. The case when code level parallelisation leads to a substantial speed-up is performing first or second order derivatives estimation, as in (A.5).

Parallelisation at method level is applied practically only for direct search non-gradient routines. Explicit sequential nature of SLP, SQP or GRG routines effectively prevents creation of their concurrent versions, at least to the author's knowing. The similar is observed in the case of conjugate directions routines (e.g. the Powell method), although the possibility of their parallelisation is still an open question [105]. Concurrent implementations of non-gradient direct search routines abound, both synchronous and asynchronous. In methods like CRS and COMPLEX the parallelisation is customarily achieved by simultaneously reflecting more than one point in the pool (cf. Section 5.1 for two concurrent CRS variants, see [17] for a description of COMPLEX with the reflecting scheme similar to the one in CRS in waveguide problem). Parallelisation of EA seems so straightforward that it should possibly be classified to code level, were it not for asynchronous implementations that require more attention and programming effort.

Parallelisation at problem level results in deep changes of the original algorithm, which are the effect of tight adaptation of the algorithm to a problem of well known and particular structure. The process starts usually with problem decomposition into subproblems, and the final algorithm structure (computing threads, synchronisation) corresponds to that decomposed structure. An exemplary problem requiring problem level parallelisation, given on p. 12, is the direct-method optimisation of a performance index of special structure.

In most simulation-optimisation problems with the simulator assumed to be a black box such far-reaching assumptions about $f(\cdot)$ and simulation partitioning are impossible. Therefore, the steam goes to construct routines explicitly exploiting the parallelism, and openly attacking the problem with sheer force. For example, Torczon [32] develops a series of deeply changed Nelder-Mead procedures where, instead of one point being reflected, the whole simplex is subject to reflection, once or more times. In the latter case, a whole lattice of trial points is generated, being the result of all anticipated reflections, expansions and contractions, and $f(\cdot)$ is evaluated concurrently for all of those points. Yet another idea [51], called parallel pattern search, consists in evaluating $f(\cdot)$ at several points in the neighbourhood of $\mathbf{x}_k$, and accepting the best of them as $\mathbf{x}_{k+1}$. Those points, unlike in the case of simulated annealing, are chosen deterministically.

B.2 Distributed processing

Closeness of the simulation module, relatively small effort consumption by the optimisation routine, flat and relatively small data structures passed to and from the simulator — all these factors determine the typical architecture of a distributed simulation-optimisation system. Simulators are considered to be servers, while the optimisation routine — run on a single machine as there is no reason to distribute the optimisation code — is the client. Simplicity of data constructs (both simulation input and output are just vectors of floating-point numbers) passed between client and servers make data marshalling easy. The passed data are few and their transmission times can be neglected as compared with average simulation time. It can be said that establishing communication using system sockets is enough.

Generally, the above is true. However, utilisation of one of several available tools facilitating distributing computations is advisable for at least three reasons. The first is data portability. It may happen to run optimisation in a heterogenous network with, say, both big-endian and little-endian machines. Manual controlling of numbers encoding is error-prone in such case. Similarly, programming the socket interface directly is considered laborious and clumsy, and therefore is avoided — this is the second reason. The third reason is that some tools offer programming interfaces well fitting the language simulator or optimisation modules are written in, thus saving a programmer the data translation job.

Six candidate tools for distributed computing have been selected to be presented here. They are:

Remote Procedure Call (RPC). A well-established mechanism that lets invoke a routine provided by a module residing on a remote machine. It comes with tools creating client-side proxies and server-side skeletons. Its standard for external representation of data structures, *XDR*, has been adopted by many later tools.

Parallel Virtual Machine (PVM). Message-passing oriented communication system making possible for processes on different machines to exchange messages. Uses *XDR* for data structures serialisation.

Message Passing Interface (MPI). Similar to PVM but focused on emulation of a parallel virtual machine with SIMD architecture.

Common Object Request Broker Architecture (CORBA). Can be presented as RPC that has upgraded to object-oriented level. CORBA's strength are many auxiliary services e.g. for message passing, synchronisation etc.

Distributed Component Object Model (DCOM). Microsoft proprietary alternative for CORBA, an object-oriented inter-process distributed communication.

Java RMI . Java native mechanism for invocation of methods provided by objects resident on another Java virtual machine. Similar in operation to CORBA and DCOM.

feature\tool	**RPC**	**PVM**	**MPI**	**CORBA**	**DCOM**	**RMI**
portability	portable	portable (widely ported)	portable†	portable	portable‡	portable
primary communication model*	call	message	message	call	call	call
interfaced languages	C	C, Fortran	C, Fortran	C, C++, Java, Smalltalk	C++, Java, Visual Basic	Java
difficulty of use	very simple	simple	simple	difficult	difficult	very simple

Table B.1: Classification of selected programming tools by their possession of attributes that are relevant in context of simulation-optimisation distributed implementation.

† Within a particular implementation. ‡ Only among Microsoft solutions. * The interprocess communication model promoted and particularly addressed by the tool. (Using the other communication model is either supported by that tool directly, or can be worked around.)

The detailed description of the above tools (except for DCOM) and of their applicability for distributed computation can be found in [118] and in [59], the latter being the author's concise review of object-oriented tools, enriched with examples and performance tests.

Features of those tools that are interesting from our point of view are summarised in Table B.1. As a conclusion, it can be said that the choice of a particular tool will probably be most dependent on that tool interfacing abilities. As an example, one can imagine a simulator required to be run under *MS-Windows* on one side (e.g. *QW-3D*), and some optimisation routine required to be run under *UNIX* (e.g. *OptdesX*). In such case RPC, PVM, MPI and

CORBA would be considered. However, in the next stage of selection MPI and CORBA would probably get excluded for their unnecessary complexity.

The issue of load balancing in order to yield as much performance as possible from a given distributed computing resources can be approached in two ways. The first approach is to write a load manager which, being simultaneously the proxy for simulation, is to decide where the currently requested simulation is to be run. Such solution leaves much control over the whole system to the user and requires no modifications of the existing simulation servers — but it implies much programming work for the load manager creation. The second approach makes use of clustering technique. A cluster, or a group of computers appearing to user and to processes as a single system, imitates a powerful parallel machine with all the implications: no extra tools for client-server data transmission are required, and the simulation-optimisation structure can be a multithreaded optimisation module spawning simulation modules locally. Next, those simulators can be transferred in the middle of their run to less loaded cluster machines. This is done at operation system kernel level and passes unnoticed. Similarly to the user-defined load balancing, clustering does not require any changes of the simulator — but it constraints substantially inter-process communication and use of system resources by the migrated processes. It also assumes cluster machines homogeneity w.r.t. the operating system. Exemplary cluster solutions are *MOSIX* [76] for *Linux* and proprietary Microsoft products [52].

Bibliography

[1] M. M. Ali and A. Törn. Population set based global optimization algorithms: Some modifications and numerical studies. *Computers & Operations Research*, 2003. Accepted for print. Available from `http://www.ima.umn.edu/preprints/feb2003/1907.pdf`.

[2] AMPL Modeling Language for Mathematical Programming (web site). Available at `http://www.ampl.com`.

[3] S. Andradóttir. A review of simulation optimization techniques. In D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 151–158.

[4] S. Andradóttir. A stochastic approximation with varying bounds. *Operations Research*, 43(6):1037–1048, November–December 1995.

[5] S. Andradóttir. A global search method for discrete stochastic optimization. *SIAM J. Optimization*, 6(2):513–530, May 1996.

[6] S. Andradóttir. Simulation optimization. In J. Banks, editor, *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. John Wiley & Sons, Inc., 1998.

[7] J. Arabas. *Wykłady z algorytmów ewolucyjnych*. Wydawnictwa Naukowo-Techniczne, 2001.

[8] J. Arabas and P. Miazga. Computer aided design of a layout of planar circuits by means of evolutionary algorithms. *Journal of Computing and Information Technology*, 7(1):61–76, 1999.

[9] P. Arabas, M. Kamola, and K. Malinowski. IP services market: Modelling, research and reality. In W. Burakowski, B. Koch, and A. Bęben, editors, *Architectures for Quality of Service in the Internet*, pages 76–87. Springer-Verlag, 2003.

[10] P. Arabas, M. Kamola, K. Malinowski, and M. Małowidzki. Pricing for IP network and services. *Information Knowledge Systems Management*, 3(2-4):153–171, 2002.

[11] A. C. Atkinson and A. N. Donev. *Optimum experimental design.* Clarendon Press, Oxford, 1992.

[12] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* Institute of Physics Publishing and Oxford University Press, 1997.

[13] M. H. Bakr, J. W. Bandler, N. Georgieva, and K Madsen. A hybrid aggressive space mapping algorithm for EM optimization. *IEEE Transactions on Microwave Theory and Techniques*, MTT-47:2440–2449, 1999.

[14] M. R. P. Barretto, L. Chwif, T. Eldabi, and R. J. Paul. Simulation optimization with the linear move and exchange move optimization algorithm. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, editors, *Proceedings of the 1999 Winter Simulation Conference*, pages 806–811.

[15] R. R. Barton and J. S. Ivey, Jr. Modifications of the Nelder-Mead simplex method for stochastic simulation response optimization. In B. L. Nelson, W. D. Kelton, and G. M. Clark, editors, *Proceedings of the 1991 Winter Simulation Conference*, pages 945–953.

[16] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming. Theory and Algorithms.* John Wiley & Sons, Inc., 1993.

[17] W. E. Biles and J. P. Kleijnen. A Java-based simulation manager for optimization and Response Surface Methodology in multiple-response parallel simulation. In P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, editors, *Proceedings of the 1999 Winter Simulation Conference*, pages 513–517.

[18] J. P. Błaszczyk. Obiektowa biblioteka do rozwiązywania zadań sterowania optymalnego z czasem dyskretnym. Master's thesis, Warsaw University of Technology, 2000.

[19] J. Boesel, R. O. Bowden, Jr., F. Glover, J. P. Kelly, and R. Westwig. Future of simulation optimization. In B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, editors, *Proceedings of the 2001 Winter Simulation Conference*, pages 1466–1469.

[20] M. J. Box. A new method of constrained optimization and a comparison with other methods. *The Computer Journal*, 8(1):42–52, April 1965.

[21] T. Brady and B. McGarvey. Heuristic optimization using computer simulations: A study of staffing levels in a pharmaceutical manufacturing laboratory. In D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 1423–1428.

[22] A. Brooke, D. Kendrick, and A. Meeraus. *Gams: A User's Guide, Release 2.25.* Boyd & Fraser Publishing Company, Danvers, 1992.

[23] A. A. Bulgak and J. L. Sanders. Integrating a modified simulated annealing algorithm with the simulation of a manufacturing system to optimize buffer sizes in automatic assembly systems. In M. Abrams, P. Haigh, and J. Comfort, editors, *Proceedings of the 1988 Winter Simulation Conference*, pages 684–690.

[24] Y. Carson and A. Maria. Simulation optimization: Methods and applications. In S. Andradóttir, K. J. Healy, and B. L. Nelson, editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 118–126.

[25] M. Celuch-Marcysiak, W. Gwarek, P. Miazga, M. Sypniewski, and A. Więckowski. Automatic design of high frequency structures using a 3-D FD TD simulator in an optimisation loop. In *XIII International Conference on Microwaves, Radar and Wireless Communications MIKON-2000*, pages 271–274, Wrocław, May 2000.

[26] H. Chen and Y. Zhu. Stochastic approximation procedures with randomly varying truncations. *Scientia Sinica*, 29(9):914–926, September 1986.

[27] E. K. P. Chong and P. J. Ramadge. Optimization of queues using an infinitesimal perturbation analysis-based stochastic algorithm with general update times. *SIAM J. Control and Optimization*, 31(3):698–732, May 1993.

[28] Software architecture and functional specifications (Pricing Module). Project deliverable D3.1.1, QOSIPS consortium, 2001.

[29] Research report on how pricing model, learning and optimisation are to be implemented to satisfy requirements. Project deliverable D3.2.1, QOSIPS consortium, 2001.

[30] D. D. Daberkow. *A Formulation of Metamodel Implementation Processes for Complex Systems Design.* PhD thesis, Georgia Institute of Technology, 2002. Available from `http://www.asdl.gatech.edu/staff/pdf/daberkow_thesis.pdf`.

[31] B. Delyon and A. Juditsky. Accelerated stochastic approximation. *SIAM J. Optimization*, 3(4):868–881, November 1993.

[32] J. E. Dennis and V. Torczon. Direct search methods on parallel machines. *SIAM J. Optimization*, 1(4):448–474, November 1991.

[33] P. D. Domański, J. Arabas, K. Świrski, and M. Wegnerowska. Economic load dispatch for combined cycle cogeneration facility: Comparison of different approaches. In *International Conference on Automation, Control and Information Technology (ACIT'2002)*, pages 130–135, Novosibirsk.

[34] Vanderplaats Research & Development, Inc. (web site). Available at `http://www.vrand.com`.

[35] A. S. Drud. GAMS/CONOPT. In *GAMS - The Solver Manuals.* GAMS Development Corporation, Washington, 1993.

[36] T. F. Edgar and D. M. Himmelblau. *Optimization of Chemical Process.* McGraw-Hill, 1988.

[37] J. F. Faccenda and R. F. Tenga. A combined simulation/optimization approach to process plant design. In J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, editors, *Proceedings of the 1992 Winter Simulation Conference*, pages 1256–1261.

[38] K. Fall and K. Varadhan. *The* ns *Manual.* UC Berkeley, 2003. Available from `http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf`.

[39] M. K. H. Fan, A. L. Tits, J. Zhou, L.-S. Wang, and J. Koninckx. CONSOLE User's Manual. Technical Research Report TR 1987-212, Systems Research Center, Univ. of Maryland, 1987.

[40] W. Findeisen, F. N. Bailey, M. Brdyś, K. Malinowski, P. Tatjewski, and A. Woźniak. *Control and Coordination in Hierarchical Systems.* John Wiley & Sons, Inc., 1980.

[41] W. Findeisen, J. Szymanowski, and A. Wierzbicki. *Teoria i metody obliczeniowe optymalizacji.* Państwowe Wydawnictwo Naukowe, 1980. In Polish.

[42] B. L. Fox and G. W. Heine. Probabilistic search with overrides. *The Annals of Applied Probability*, 5(4):1087–1094, 1995.

[43] M. C. Fu. A tutorial review of techniques for simulation optimization. In J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, editors, *Proceedings of the 1994 Winter Simulation Conference*, pages 149–156.

[44] M. C. Fu. Convergence of a stochastic approximation algorithm for the G1/G/1 queue using infinitesimal perturbation analysis. *Journal of Optimization Theory and Applications*, 65(1):149–160, April 1990.

[45] S. B. Gelfand and S. K. Mitter. Simulated annealing with noisy or imprecise energy measurements. *Journal of Optimization Theory and Applications*, 62(1):49–62, July 1989.

[46] S. B. Gelfand and S. K. Mitter. Recursive stochastic algorithms for global optimization in $\mathcal{R}^d$. *SIAM J. Control and Optimization*, 29(5):999–1018, September 1991.

[47] F. Glover, J. P. Kelly, and M. Laguna. New advances and applications of combining simulation and optimization. In J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, editors, *Proceedings of the 1996 Winter Simulation Conference*, pages 144–152.

[48] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[49] W. Gwarek and P. Miazga. Improved design of coaxial impedance transformers using electromagnetic 2-D solver in an optimization loop. In *XI International Conference on Microwaves, Radar and Wireless Communications MIKON-1996*, pages 433–437, Warszawa, May 1996.

[50] U. Hammel. Chapter F.1.8: Simulation models. In T. Bäck, D. B. Fogel, and Z. Michalewicz, editors, *Handbook of Evolutionary Computation*. Institute of Physics Publishing, Bristol, 1997.

[51] P. D. Hough and T. G. Kolda. Asynchronous parallel pattern search for nonlinear optimization. *SIAM J. Scientific Computing*, 23(1):134–156, June 2001.

[52] High Performance Computing for Windows Server 2003. Web page available from `http://www.microsoft.com/windowsserver2003/hpc`.

[53] HPF: The High Performance Fortran home page (web site). Available at `http://www.crpc.rice.edu/HPFF`.

[54] J. Hromkovič. *Algorithmics for Hard Problems*. Springer-Verlag, 2001.

[55] D. G. Humphrey and J. R. Wilson. A revised simplex search procedure for stochastic simulation response-surface optimization. In D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 751–759.

[56] Optimizing applications with the Intel® C++ and Fortran compilers for Windows and Linux. Product leaflet, Intel Corporation.

[57] S. H. Jacobson. Convergence results for harmonic gradient estimators. *ORSA Journal on Computing*, 6:381–397, 1994.

[58] Z. Jankowski, Ł. Kurpisz, L. Laskowski, J. Łajkowski, A. Miller, W. Sikora, J. Portacha, and M. Zgorzelski. Model matematyczny pracy turbozespołu w zmiennych warunkach — na przykładzie bloku 200 MW. *Biuletyn Informacyjny Instytutu Techniki Cieplnej*, 33:3–36, 1972. In Polish (English abstract available).

[59] M. Kamola. Rozdział 7: Narzędzia do tworzenia obiektowych programów rozproszonych w środowiskach sieciowych. In A. Karbowski and E. Niewiadomska-Szynkiewicz, editors, *Obliczenia równoległe i rozproszone*. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2001.

[60] M. Kamola, P. Arabas, and K. Malinowski. Uncertainty in modelling and its impact on optimisation domain; network services pricing. In *Proceedings of Summer Computer Simulation Conference (SCSC) 2003*, pages 490–496, Montreal.

[61] M. Kamola and K. Malinowski. Simulator-optimizer approach to planning of plant operation; ill-defined simulator case. In P. P. Groumpos and A. P. Tzes, editors, *A Proceedings volume from the IFAC Symposium, Patras Greece, 12-14 July 2000*.

[62] A. Karbowski, K. Malinowski, and E. Niewiadomska-Szynkiewicz. Rozdział 8: Algorytmy synchroniczne. In A. Karbowski and E. Niewiadomska-Szynkiewicz, editors, *Obliczenia równoległe i rozproszone*. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2001.

[63] N. L. Kleinman, S. D. Hill, and V. A. Ilenda. Simulation optimization of air traffic delay cost. In D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivannan, editors, *Proceedings of the 1998 Winter Simulation Conference*, pages 1177–1181.

[64] P. Kozakowski. *Analiza czasowa pasywnych układów mikrofalowych o dużej dobroci*. PhD thesis, Politechnika Gdańska, Gdańsk, 2002. Available from `http://www.pg.gda.pl/mwave-mim/THESES/piotek_phd.pdf`.

[65] H. J. Kushner and J. Yang. Stochastic approximation with averaging of the iterates: Optimal asymptotic rate of convergence for general processes. *SIAM J. Control and Optimization*, 31(4):1045–1062, July 1993.

[66] D. A. Malinowska. Steady-state optimisation of high pressure gas networks. Master's thesis, Warsaw University of Technology, March 1996.

[67] K. Malinowski. Optimization network flow control and price coordination with feedback: proposal of a new distributed algorithm. *Computer Communications*, 25:1028–1036, 2002.

[68] The MathWorks, Inc. *Optimization Toolbox User's Guide*. Available from `http://www.mathworks.com/access/helpdesk/help/pdf_doc/optim/optim_tb.pdf`.

[69] C. C. Meewella and D. Q. Mayne. Efficient domain partitioning algorithms for global optimization of rational and Lipschitz continuous functions. *Journal of Optimization Theory and Applications*, 61(2):247–270, May 1989.

[70] S. Meinzer, A. Quinte, M. Gorges-Schleuter, W. Jakob, and W. Süß. Simulation and design optimization of microsystem based on standard simulators and adaptive search techniques. In *Proceedings of European Design Automation Conference with EURO-VHDL'96 and exhibition on European Design Automation*, pages 322–327, Geneva, 1996.

[71] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[72] P. Miazga. Optimization of the microstrip to waveguide transition. Application note, Institute of Radioelectronics, Warszawa, 2001.

[73] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.

[74] A. Miller and J. Lewandowski. Wyznaczanie charakterystyk układów z turbinami cieplnymi. *Zeszyty naukowe Politechniki Łódzkiej*, 89(389):161–168, 1981. In Polish (English abstract available).

[75] A. Miller and J. Lewandowski. *Praca turbin parowych w zmienionych warunkach*. Wydawnictwa Politechniki Warszawskiej, Warszawa, 1992. In Polish.

[76] MOSIX (web site). Available at `http://www.mosix.org`.

[77] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

[78] NEOS Server for Optimization (web site). Available at `http://www-neos.mcs.anl.gov/neos`.

[79] OpenMP: Simple, portable, scalable SMP programming (web site). Available at `http://www.openmp.org`.

[80] OPNET Technologies, Inc. *OPNET Modeler. Accelerating Network R&D.* Product brochure available from `http://www.opnet.com/products/modeler/opnet_modeler.pdf`.

[81] OptTek Systems, Inc. (web site). Available at `http://www.opttek.com`.

[82] P. Y. Papalambros. *Principles of optimal design.* Cambridge University Press, 1988.

[83] A. Parkinson, R. Balling, J. Free, J. Talbert, D. Davidson, G. Gritton, L. Borup, and B. Busaker. *OptdesX^{TM}; A Software System for Optimal Engineering Design. Users Manual. Release 1.0.* Design Synthesis, Inc., Provo, 1992.

[84] PAS, Inc. (web site). Available at `http://www.pas.com`.

[85] R. Penrose. *The Emperor's New Mind.* Oxford University Press, 1999.

[86] G. Ch. Pflug. *Optimization of Stochastic Models.The Interface between Simulation and Optimization.* Kluwer Academic Publishers, 1996.

[87] E. L. Plambeck, B.-R. Fu, S. M. Robinson, and R. Suri. Sample-path optimization of convex stochastic performance methods. *Mathematical Programming*, 75:137–176, 1996.

[88] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control and Optimization*, 30(4):838–855, July 1991.

[89] J. Portacha. *Optymalizacja struktury układu cieplnego siłowni parowych.* PhD thesis, Politechnika Warszawska, Warszawa, 1969. In Polish.

[90] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, 1992.

[91] W. L. Price. Global optimization by controlled random search. *Journal of Optimization Theory and Applications*, 40:333–348, 1983.

[92] W. L. Price. Global optimization algorithms for a CAD workstation. *Journal of Optimization Theory and Applications*, 55(1):133–146, October 1987.

[93] QWED, Ltd., Warszawa. *QuickWave-3D User's Manual*, 2003.

[94] QWED, Ltd., Warszawa. *QW-Optimizer User's Manual*, 2003.

[95] E. Radwański, R. Skowroński, and A. Twarowski. *Problemy modelowania systemów energotechnologicznych.* Instytut Techniki Cieplnej PW, Warszawa, 1993. In Polish.

[96] S. S. Rao. *Engineering Optimization. Theory and Practice.* John Wiley & Sons, Inc., 1996.

[97] S. M. Robinson. Analysis of sample path optimization. *Mathematics of Operations Research*, 21:513–528, 1996.

[98] Rogue Wave Software, Inc. (web site). Available at `http://www.roguewave.com`.

[99] D. Ruppert. A Newton-Raphson version of the multivariate Robbins-Monro procedure. *The Annals of Statistics*, 13(1):236–245, 1985.

[100] R. E. Shannon. Introduction to simulation. In J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson, editors, *Proceedings of the 1992 Winter Simulation Conference*, pages 65–73.

[101] A. Shapiro and Y. Wardi. Convergence analysis of gradient descent stochastic algorithms. *Journal of Optimization Theory and Applications*, 91(2):439–454, November 1996.

[102] A. Shapiro and Y. Wardi. Convergence analysis of stochastic algorithms. *Mathematics of Operations Research*, 21(3):615–628, August 1996.

[103] H. Simon. *Price Management.* Elsevier Science Publishers, 1989.

[104] M. G. Singh and J-C. Bennavail. Price-Strat. A knowledge support system for profitable decision-making during price wars. *Information and Decision Technologies*, 19:277–296, 1994.

[105] J. Sobczyk and A. P. Wierzbicki. Pulsar algorithms: A class of coarse-grain parallel nonlinear optimization algorithms. Working Paper WP-94-53, International Institute for Applied Systems Analysis, 1994.

[106] SOCS Capabilities. Web page available from `http://www.boeing.com/phantom/socs/capabilities.html`.

[107] J. C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, March 1992.

[108] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4(4):441–461, November 1962.

[109] B. Stuckman, G. Evans, and M. Mollaghasemi. Comparison of global search methods for design optimization using simulation. In B. L. Nelson, W. D. Kelton, and G. M. Clark, editors, *Proceedings of the 1991 Winter Simulation Conference*, pages 937–944.

[110] A Svensson. A novel simulation based optimization method - applied to the dimensioning of circuit switched networks. In *Proceedings of the 1st World Congress on Systems Simulation*, September 1997.

[111] Synaps, Inc., Atlanta. *Epogy 2003. User's Guide.*

[112] Synaps, Inc. *Synaps Pointer — Optimize.* Web page available from `http://www.synaps-inc.com/products/pointer/optimize/index.html`.

[113] A. Taflove. *Computational Electrodynamics: The Finite-Difference Time-Domain Method.* Artech House, Boston, 1995.

[114] K. Urbaniec. Określanie stanu układu cieplnego bloku elektrowni jądrowej metodą minimalizacji. *Archiwum Energetyki*, 1-2:3–12, 1979. In Polish (English abstract available).

[115] D. Verma. *Supporting Service Level Agreements on IP Networks.* Macmillan Technical Publishing, Indianapolis, 1999.

[116] F. H. Walters, Jr., L. R. Parker, S. L. Morgan, and S. N. Deming. *Sequential simplex optimization: a technique for improving quality and productivity in research, development and manufacturing.* CRC Press, Inc., 1991.

[117] M. Warchoł, A. Karbowski, and E. Niewiadomska-Szynkiewicz. Rozdział 5: Narzędzia do programowania równoległego na maszynach wieloprocesorowych z pamięcią wspólną. In A. Karbowski and E. Niewiadomska-Szynkiewicz, editors, *Obliczenia równoległe i rozproszone.* Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2001.

[118] M. Warchoł, A. Karbowski, and E. Niewiadomska-Szynkiewicz. Rozdział 6: Narzędzia do programowania na maszynach z pamięcią lokalną oraz w sieciach komputerowych.

In A. Karbowski and E. Niewiadomska-Szynkiewicz, editors, *Obliczenia równoległe i rozproszone*. Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa, 2001.

[119] Y. Wardi. Stochastic algorithms with Armijo stepsizes for minimization of functions. *Journal of Optimization Theory and Applications*, 64(2):399–417, February 1990.

[120] C. Z. Wei. Multivariate adaptive stochastic approximation. *The Annals of Statistics*, 15(3):1115–1130, 1987.

[121] D. Yan and H. Mukai. Stochastic discrete optimization. *SIAM J. Control and Optimization*, 30(3):594–612, May 1992.

[122] D. Yan and H. Mukai. Optimization algorithm with probabilistic estimation. *Journal of Optimization Theory and Applications*, 79(2):345–371, November 1993.

[123] G. Yin. On extensions of Polyak's averaging approach to stochastic approximation. *Stochastics and Stoschastics Reports*, 36:245–264, 1991.

List of Figures

List of Tables