TR3524-S

STELLINGEN

behorende bij het proefschrift

Application of Mathematical Optimization Techniques to Nuclear Reactor Reload Pattern Design

door A.J. Quist

I

Het model met gescheiden leeftijdsfracties in bundels biedt voldoende perspectieven voor een vervolgonderzoek.

Dit proefschrift, par 2.3.3 en 8.3.

II

Locale zoekmethoden zijn succesvol bij het oplossen van een groot aantal combinatorische optimaliseringsproblemen. Bij problemen waarvan de moeilijkheid mede bepaald wordt door niet-lineaire relaties op de variabelen, kan het beter zijn om deze methoden te combineren met niet-lineaire optimalisatiemethoden.

Dit proefschrift, par 8.2.

III

Vergelijking van methoden voor optimalisatie van herlaadpatronen wordt ernstig belemmerd door het ontbreken van een algemeen geaccepteerde verzameling van testproblemen, zoals die bijvoorbeeld bestaat voor een groot aantal probleemklassen binnen de mathematische optimalisering.

Dit proefschrift, par 3.4.

IV

Het *NP*-complete probleem

$$\text{minimize}\{x^T A x : \; x \in \mathbb{R}^n, \; \sum_i x_i = 1, \; x \geq 0\}$$

is te herleiden tot het volgende convexe optimalisatieprobleem:

$$\text{minimize}\{\mathrm{Tr}(AX) : \; X \in \boldsymbol{B}, \; \mathrm{Tr}(X) = 1\}.$$

Hierin is $\boldsymbol{B}$ de (convexe) kegel van completely positive $n \times n$-matrices.

I.M. Bomze, M. Dür, E. de Klerk, C. Roos, A.J. Quist and T. Terlaky, On copositive programming and standard quadratic optimization problems, te verschijnen in *Journal of Global Optimization*

V

Technisch onderzoek uit het volledige onderzoeksveld van de TU Delft kan baat hebben bij combinatie van de beschikbare kennis over het numeriek oplossen van differentiaalvergelijkingen en de beschikbare kennis over wiskundige optimalisatie.

Voorbeeld: A.H. Lobbrecht, *Dynamic Water-System Control: Design and Operation of Regional Water-Resources Systems,* Proefschrift, TU Delft, 1997

VI

Interdisciplinair onderzoek verkrijgt meerwaarde indien de samenwerkende onderzoekers één of meer conferenties uit de andere disciplines bezoeken.

VII

Recentelijk is het een promovendus gelukt om maar liefst 72 stellingen bij zijn proefschrift te voegen. Zowel aantal als inhoud bleven echter ver achter bij de stellingen van Salomo, zoals neergelegd in zijn boek der Spreuken.

A. van Dijk, *Aliasing in one-point turbulence measurements: Theory, DNS and hotwire experiments,* Proefschrift, TU Delft, 1999

VIII

> 'De beweringen in 1 Koningen 7,23 en 2 Kronieken 4,2 van het Oude Testament zijn onjuist.'
> Stelling 7 bij het proefschrift *The Adjoint of a Semigroup of Linear Operators,*
> J.M.A.M. van Neerven, 1992, CWI, Amsterdam.

De betrouwbaarheid van de Bijbel als Woord van God staat of valt niet met een afrondingsfout.

IX

Een eventueel besluit om de kerncentrale Borssele in 2004 alsnog te sluiten, is moeilijk te rijmen met invoer van elektriciteit uit het buitenland.

X

Een hedendaagse promovendus kan internet niet meer missen.

XI

Reclame en media willen ons ten onrechte laten geloven dat bijna niemand zonder internet kan.

XII

Een van de raadgevingen voor het geval er brand uitbreekt, is om te gaan kruipen in verband met de rookontwikkeling. Het verdient daarom aanbeveling om bordjes 'Nooduitgang' op kniehoogte te bevestigen.

XIII

Aggressieve verkeersovertreders kunnen hun gedrag alleen volhouden in de veronderstelling dat anderen zich wel aan de regels houden.

XIV

Haast zonder humor leidt tot stress.

PROPOSITIONS

belonging to the thesis

Application of Mathematical Optimization Techniques to Nuclear Reactor Reload Pattern Design

by A.J. Quist

I

The model with separate age fractions in the bundles opens sufficient perspectives for follow-up research.

This thesis, par 2.3.3 and 8.3.

II

Local search methods are successful for solving a large number of combinatorial optimization problems. For problems where the difficulty is also due to nonlinear relations between the variables, it can be advantageous to combine those methods with techniques from nonlinear optimization.

This thesis, par 8.2.

III

Comparison of methods for reload pattern optimization is hampered by the lack of a generally accepted set of test problems. Such test sets do for example exist for a large number of problem classes within mathematical optimization.

This thesis, par 3.4.

IV

The ***NP***-complete problem

$$\text{minimize}\{x^T A x:\ x \in \mathbb{R}^n,\ \sum_i x_i = 1,\ x \geq 0\}$$

can be reformulated as the convex optimization problem

$$\text{minimize}\{\text{Tr}(AX):\ X \in \boldsymbol{B},\ \text{Tr}(X) = 1\}.$$

Here, $\boldsymbol{B}$ is the (convex) cone of completely positive $n \times n$-matrices.

I.M. Bomze, M. Dür, E. de Klerk, C. Roos, A.J. Quist and T. Terlaky, On copositive programming and standard quadratic optimization problems, to appear in *Journal of Global Optimization*

V

Researchers from the entire spectrum of engineering research activities at the Delft University of Technology can benefit from combining the available expertise on the numerical solution of differential equations and mathematical optimization.

Example: A.H. Lobbrecht, *Dynamic Water-System Control: Design and Operation of Regional Water-Resources Systems*, PhD thesis, Delft University of Technology, 1997

VI

Interdisciplinary research benefits when researchers attend conferences outside their own field of speciality.

VII

Recently, a PhD student presented 72 propositions with his thesis. This falls far short of the propositions of Solomon in his book of Proverbs, both in quantity and profundity.

A. van Dijk, *Aliasing in one-point turbulence measurements: Theory, DNS and hotwire experiments*, PhD thesis, Delft University of Technology, 1999

VIII

> 'The statements in 1 Kings 7,23 and 2 Chronicles 4,2 of the Old Testament are incorrect.'
> Proposition 7 by the PhD thesis *The Adjoint of a Semigroup of Linear Operators*,
> J.M.A.M. van Neerven, 1992, CWI, Amsterdam.

A rounding error has no bearing on the reliability of the Bible as the Word of God.

IX

A possible decision to close the Borssele nuclear reactor in 2004, cannot be reconciled with the import of electricity.

X

A contemporary PhD student cannot function without internet.

XI

Advertisements and the media wrongly suggest that almost nobody can function without internet.

XII

It is advised to crawl in case of fire, to prevent smoke inhalation. Therefore, it is advisable to place exit signs at knee-height.

XIII

Aggressive traffic offenders can only keep up their behavior by assuming that other people obey the traffic laws.

XIV

Haste leads to stress in the absence of humor.

3524
739647
3058297
TR3524

Application of Mathematical Optimization Techniques to Nuclear Reactor Reload Pattern Design

A.J. Quist

Application of

Mathematical Optimization Techniques

to Nuclear Reactor Reload Pattern Design

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof.ir. K.F. Wakker,
in het openbaar te verdedigen ten overstaan van een commissie,
door het College voor Promoties aangewezen,
op maandag 29 mei 2000 te 16:00 uur
door

Adriaan Jacobus QUIST

wiskundig ingenieur
geboren te Tholen

Dit proefschrift is goedgekeurd door de promotoren:
Prof.dr.ir. C. Roos
Prof.dr. F.A. Lootsma

Toegevoegd promotor:
Dr.ir. J.E. Hoogenboom

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof.dr.ir. C. Roos	Universiteit Leiden, promotor
Prof.dr.ir. F.A. Lootsma	Technische Universiteit Delft, promotor
Dr.ir. J.E. Hoogenboom	Technische Universiteit Delft, toegevoegd promotor
Prof.dr. T. Terlaky	McMaster University, Ontario, Canada
Prof.dr.ir. H. van Dam	Technische Universiteit Delft
Prof.dr. W.K. Klein Haneveld	Rijksuniversiteit Groningen
Prof.dr. N.V. Sahinidis	University of Illinois, USA

Dit proefschrift kwam tot stand onder auspiciën van:

THOMAS STIELTJES INSTITUTE
FOR MATHEMATICS

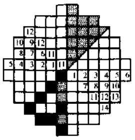
The cover illustration shows the objective function of the basic model and the objective function of the new model as developed in Section 2.3.3, and is explained on 138.

ISBN: 90-9013770-X

© A.J. Quist, Delft 2000.

All rights reserved. No part of this publication may be reproduced in any form or by any electronic or mechanical means including information storage and retrieval systems without the prior written permission from the author.

Printed by Universal Press



Contents

List of Figures v

List of Tables vii

List of Algorithms ix

Summary xi

Samenvatting xiii

1 Introduction **1**
- 1.1 Nuclear reactor fuel management 1
- 1.2 Mathematical optimization 6
  - 1.2.1 Historical overview and basics 6
  - 1.2.2 More complicated models 7
  - 1.2.3 Example application 8
- 1.3 Motivation, aims and scope 11
- 1.4 Outline 11

2 Problem statement **13**
- 2.1 Reactor Physics 14
  - 2.1.1 Fission chain reactions 15
  - 2.1.2 Neutron transport 17
  - 2.1.3 One and a half group diffusion theory 19
  - 2.1.4 Green function model 23
  - 2.1.5 Fuel burnup 24
  - 2.1.6 Burnable absorbers 26
  - 2.1.7 Different types of nuclides 28
  - 2.1.8 Other extensions 29
- 2.2 Elements of the optimization models 31
  - 2.2.1 Loading pattern specification 31
  - 2.2.2 Equilibrium cycle condition 32
  - 2.2.3 Quadrant and Octant symmetry 33
  - 2.2.4 Safety restriction 35
  - 2.2.5 Objective function 35
- 2.3 Summary of the basic models 35
  - 2.3.1 The basic $1^{1}/_{2}$ group model 36

2.3.2 Model including Burnable Poisons 38
2.3.3 A new model 39

3 Existing optimization approaches **41**
3.1 Direct search methods 42
3.1.1 Pairwise or Multiple Interchange 43
3.1.2 Simulated Annealing 45
3.1.3 Evolutionary algorithms 46
3.1.4 Tabu search 47
3.1.5 Global to local search 48
3.1.6 Other methods in literature 49
3.2 Continuous optimization approaches 49
3.3 Tools for general use 52
3.3.1 Perturbation theory (PT) 52
3.3.2 Engineering constraints 52
3.4 Conclusion 53

4 Mixed-integer nonlinear optimization **55**
4.1 General structure of MINLP 55
4.2 The nonlinear part 57
4.2.1 Basics of nonlinear optimization theory 59
4.2.2 Unconstrained optimization methods 62
4.2.3 Obtaining derivative information 64
4.2.4 Constrained optimization methods. 65
4.3 The integer part 69
4.3.1 Rounding heuristics 71
4.3.2 The Branch-and-Bound method 72
4.3.3 Outer Approximation 74
4.3.4 Cutting Planes 76
4.3.5 Other techniques 78
4.4 Global optimization approaches 79
4.4.1 Lipschitz optimization 79
4.4.2 Convex relaxations within Branch and Reduce 80
4.5 Summary 82

5 Implementation issues in mixed-integer nonlinear optimization **85**
5.1 Starting point and fixed core evaluation 87
5.1.1 Initial values of the assignment variables 87
5.1.2 Initial values of the physical variables 90
5.1.3 Fixed core evaluation 90
5.2 The nonlinear part 92
5.2.1 Variable bounds 93
5.2.2 Addition of constraints 95
5.2.3 Constraint relaxation 95
5.2.4 Engineering constraints 97

5.2.5 Reducing nonconvexity by reformulations . . . 98
5.2.6 Time-discretization . . . 100
5.3 The integer part . . . 102
5.3.1 Rounding heuristics . . . 103
5.3.2 Outer approximation . . . 104
5.3.3 Branch-and-Bound . . . 105
5.3.4 Pairwise interchange . . . 107
5.3.5 Combination of methods . . . 107
5.4 Cuts . . . 108
5.4.1 Cut type 1 . . . 110
5.4.2 Cut type 2 . . . 111
5.4.3 Cut type 3 . . . 112
5.4.4 Power peak strengthening. . . . 113
5.4.5 Warm start after the addition of cuts . . . 114
5.4.6 General issues regarding the cuts . . . 116
5.5 Test environments and solvers . . . 117
5.5.1 GAMS . . . 118
5.5.2 Fortran and C codes . . . 119
5.5.3 Graphical user interface . . . 121

6 Computational results 127
6.1 The test sets . . . 128
6.2 Results for the basic models . . . 132
6.2.1 Different modeling features and modeling variants . . . 132
6.2.2 Nonlinear optimization solvers . . . 140
6.2.3 Mixed-integer solvers . . . 140
6.2.4 Cuts . . . 146
6.3 Results for the BP model . . . 147
6.4 Conclusion . . . 150

7 Alternative approaches 153
7.1 Optimization via the web . . . 153
7.2 LANCELOT and Leyffer's code . . . 154
7.3 Global optimization . . . 155
7.4 Semidefinite and copositive relaxation . . . 156
7.4.1 Semidefinite Optimization . . . 157
7.4.2 Copositive Optimization . . . 158
7.5 Parallelization . . . 159
7.5.1 Some concepts in parallelization . . . 160
7.5.2 Parallelization in neighborhood search . . . 161
7.5.3 Parallelization in MINLP . . . 162
7.5.4 Conclusion . . . 164

8 Summary, Conclusions and Recommendations **167**
8.1 Summary . . . 167
8.2 General conclusions . . . 170
8.3 Recommendations . . . 171

A Nonlinear optimization algorithms **175**
A.1 Overview of algorithms . . . 176
A.1.1 Sequential or Successive Linear Programming (SLP) . . . 176
A.1.2 Sequential Quadratic Programming (SQP) . . . 177
A.1.3 Projected Lagrangian Method (PLM) . . . 177
A.1.4 Augmented Lagrangian method (ALM) . . . 178
A.1.5 Reduced Gradient method (RGM) . . . 178
A.1.6 Generalized Reduced Gradient (GRG) method . . . 180
A.1.7 Augmented-Lagrangian Trust-region method (ALTR) . . . 181
A.1.8 Interior point methods (IPM) . . . 182
A.2 Tools for nonlinear optimization algorithms . . . 184
A.2.1 Quasi Newton updates of the (inverse) Hessian . . . 184
A.2.2 Line search methods . . . 185

B Some proofs **187**
B.1 Results on eigenvalues and eigenfunctions . . . 187
B.1.1 Discrete inverse equation (2.15) . . . 187
B.1.2 Continuous diffusion equation (2.9) . . . 188
B.2 Mathematical proof that k^∞ decreases during a cycle . . . 190

Bibliography **193**

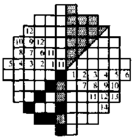
List of symbols **203**

List of abbreviations **205**

Index **207**

Dankwoord **209**

Curriculum Vitae **211**



List of Figures

1.1 Schematic picture of a normal operating nuclear power plant 2
1.2 Schematic view of a fuel loading pattern 2
1.3 One fuel element, consisting of a number of fuel pins 3
1.4 Subcritical loading pattern 5
1.5 Super-critical loading pattern 5
1.6 Boat crossing the river 9

2.1 Binding energy as function of the mass number 15
2.2 Power as function of time for each bundle, without and with burnable poisons . 27
2.3 Nuclide Chain 28
2.4 Comparison of k_{EoC}^{eff} and the maximum power peak on 34 test cores, obtained by a fast nodal method and a detailed diffusion method 30
2.5 Octant symmetric core 34
2.6 Fractional bundle reloading, wrong (left) and correct (right) 39

3.1 Pairwise Interchange example I 44
3.2 Pairwise Interchange example II and III 44

4.1 Optimization of a discrete problem 56
4.2 Optimization of two nonconvex problems 57
4.3 Illustration of the KKT conditions 60
4.4 Saddle point illustration for $f(x_1,x_2) = x_1^2 - x_2^2$ 61
4.5 Illustration of the RGM and the GRG method 67
4.6 Example Branch-and-Bound tree 73
4.7 Convex region estimated by linear inequalities 75
4.8 Small MILP example where cuts can be derived 77
4.9 Cut in nonconvex region wrongly cuts off better integer solutions 78
4.10 Underestimation of a nonconvex objective function: a) using the Lipschitz constant; b) using the convex envelope 80
4.11 Bilinear function underestimated by 2 linear functions 81

5.1 Good loading pattern with ring structure 88
5.2 Most likely locations for bundles in the different age groups 88
5.3 Tightening of linear bounds and relaxation of nonconvex constraints 95
5.4 Center of an octant symmetric core 98
5.5 Central vs. forward differences solutions as function of the number of time steps 100

5.6 Comparison of k_T^{eff} obtained by forward and central discretization using $T = 6$ and $T = 15$ . . . 101
5.7 Fractional solution due to the power peak constraint . . . 103
5.8 Results of a fixed core evaluation for 2000 patterns on test data set Small 1 . . . 109
5.9 High fluxes in surrounding nodes that influence the power in node i . . . 110
5.10 Absorption probabilities $G_{i,j}$ around node j . . . 112
5.11 Main window and the two drag and drop windows of our GUI 'PlotCore' . . . 123
5.12 Core evaluation window, and a plot showing the k^{∞} development in time for the different nodes . . . 124
5.13 Two examples of the interpolation tool in PlotCore . . . 125
5.14 Pairwise interchange windows . . . 126

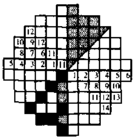
6.1 The four different core layouts . . . 129
6.2 Absorption probabilities $G_{i,j}$ in nodes i surrounding center node j . . . 130
6.3 Comparison of different starting points . . . 133
6.4 Comparison of the Basic model and the k^{EoC}-model . . . 136
6.5 Conopt+Rounding algorithm with forward and central discretization using 6 and 10 time steps per cycle . . . 137
6.6 Objective value as function of the iteration for the fractional model and test set Medium 3 . . . 139
6.7 Different CPLEX settings in DICOPT . . . 141
6.8 Results for different settings of the PI algorithm . . . 143
6.9 Results for Cuts . . . 147
6.10 Results with and without use of BP . . . 149

7.1 Speedup that is achieved by parallelization of PI . . . 163

A.1 Illustration of the Goldstein-Armijo criterion . . . 186

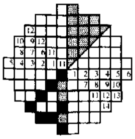
List of Tables

6.1 The various computing environments 127
6.2 Parameter values for the different test sets 130
6.3 Elements of the G-matrices in the different test sets 130
6.4 Number of variables, constraints, linear and nonlinear nonzeros for the different models 131
6.5 The effect of different model variants 135
6.6 Results of the new fractional model with Rounding algorithm 139
6.7 CONOPT vs. MINOS5. Results of the continuous optimization 140
6.8 Results on the basic model with five different algorithms 145
6.9 Effective Δ_t-values for different types of discretization with different number of time steps 148
6.10 Solution from which we started, CRPR solution and best-known solution for the Small, Medium and Large test sets 151



List of Algorithms

5.1 Core evaluation routine for fixed loading patterns 91
5.2 Burnup calculation using central discretization 92
5.3 DICOPT algorithm to solve nonlinear mixed-integer problems 104
5.4 PI algorithm exploring infeasible neighbors 108
5.5 General outline of the cut-algorithm 110



Summary

APPLICATION OF MATHEMATICAL OPTIMIZATION TECHNIQUES TO NUCLEAR REACTOR RELOAD PATTERN DESIGN

Nuclear power plants are an important source of electricity in many countries. The uranium that is used in these plants is very costly, and in order to reduce both fuel cost and nuclear waste, it should be utilized efficiently. Nuclear fuel management is concerned with the question of how to utilize the uranium in a safe and efficient way. Our research dealt with one important aspect of fuel management, namely the placement of fuel elements in the reactor core. A nuclear reactor core generally consists of a number of nuclear fuel elements, which can be divided into groups of different ages. At the end of a core operation cycle (which usually lasts from 12 up to 18 months), the oldest group is discarded from the core, the remaining fuel elements may be reshuffled in the core grid, and the empty positions are filled with new, unburned fuel elements. A scheme that tells where each element should be moved to is known as a reload pattern. The positions of the elements in the reload pattern can have big influences on the efficiency of the reactor. In current practice, such patterns are usually designed with the use of techniques that are mathematically known as direct search heuristics.

Another approach to reload pattern optimization is the use of derivative-based optimization methods. For a given reload pattern, the behavior of the reactor during the forthcoming cycle is described by a set of differential equations. These equations are discretized, which leads to a system of algebraic equations. Derivative information on these equations is easily available, and can be used in a mathematical optimization model. The purpose of our research has been to explore the usefulness of this approach, and to find the advantages and disadvantages of derivative based methods when applied to reload pattern optimization. This is done by developing and implementing appropriate mathematical optimization models and algorithms.

After the Introduction, the second chapter gives a description of the underlying physics, and the differential equations that are involved. A very accurate model, including all details of the specific core layout, cooling liquid, three spatial dimensions, and so on, would be so large, that evaluating just a single pattern would already take lots of computation time. Therefore, we have developed an optimization model that captures the essence of the physical process, so that it gives a very reasonable indication of the yield and the safety margins for a pattern, while at the same time it is small enough to be handled in an optimization context. The model as such does not require that the variables that describe the loading pattern are integer-valued, *i.e.*, fuel elements may be mixed, which is an advantage when using derivative based methods. It is the responsibility of the optimization method to find integer-valued solutions. The model is further extended to include the optimization of the burnable absorber concentration. Burnable absorbers are added to the core to damp the power of fresh bundles at the beginning of a cycle, so giving a better balanced power distribution.

Chapter 3 provides an extensive overview of methods that are applied to the reload pattern optimization problem in existing literature. Most papers show an implementation of direct search heuristics, like pairwise interchange, simulated annealing, and evolutionary algorithms. Few papers use derivative-based optimization algorithms. In these cases, the models are usually smaller and less detailed than the model we are working with. Perturbation theory is a general tool that has been used in both approaches. The same holds for the use of engineering knowledge to narrow the search area.

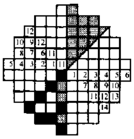
The fourth chapter lists a number of mixed-integer nonlinear optimization methods that can be useful for reload pattern design. Variants of Sequential Quadratic Programming, the Augmented Lagrangian Trust Region method and the (Generalized) Reduced Gradient methods are applied in the different implementations that we have used. Since all these algorithms necessarily assume convexity of the model, they will find local optimal solutions for our problem (which is the case for the direct search methods as well). Integer solutions are found via rounding heuristics, via a Branch-and-Bound method and with Outer Approximation. Cuts can be added to find integer solutions more efficiently, and we also used them to cut off bad local optimal solutions. Another approach is the use of global optimization.

Chapter 5 is the core chapter, describing all the issues that had to be addressed in the actual implementation. Due to the size and the nonconvexity of the problem, lots of specific adaptations, and tests with several solvers were needed to find good local optimal solutions. One critical issue was the starting point. We developed some variants of starting point generators that gave satisfactory solutions. The nonlinear optimization algorithm is further guided by tightening bounds, by addition of artificial constraints and by the initial relaxation of some critical constraints. Some more modifications have been investigated that were less successful. Furthermore, cuts have been added to escape from local optimal solutions, and new starting points are derived after the addition of these cuts. In order to guarantee stability, the time has to be discretized in a careful way. Finally, we have combined our method with a pairwise interchange direct search heuristic. In order to visualize the results, a graphical user interface has been built, in which all characteristics of a solution can be viewed, and that gives a real-time view of the pairwise interchange algorithm.

For the final tests we have used a commercially available Outer Approximation solver, and a commercially available generalized reduced gradient solver, combined with our own rounding and pairwise interchange algorithms. As is shown in Chapter 6, this last combination gives results that are competitive to solutions obtained with direct search methods only, while the computation time is much shorter. Furthermore, our model can easily be extended with continuous variables, for example to model (continuous variable) burnable poisons. This is much less evident when using direct search methods.

Finally, a short seventh chapter is devoted to miscellaneous algorithms that were considered only briefly. Among others, these include less successful nonlinear optimization software packages, global optimization methods, the use of parallelization, and a semidefinite relaxation.

Arie Quist, May 2000 Delft University of Technology



Samenvatting

TOEPASSING VAN TECHNIEKEN UIT DE WISKUNDIGE OPTIMALISATIE BIJ HET ONTWERP VAN HERLAADPATRONEN VOOR KERNREACTOREN

In veel landen spelen kerncentrales een belangrijke rol bij de energie-opwekking. Het uranium dat in deze centrales wordt gebruikt is erg kostbaar. Om deze reden, en om de hoeveelheid nucleair afval te beperken, moet er efficiënt mee worden omgegaan. Ook veiligheid speelt een grote rol. Het werk in dit proefschrift richt zich op één aspect van kernbrandstofmanagement, te weten de plaatsing van brandstofelementen in de reactorkern. Een reactorkern bevat in het algemeen een aantal splijtstofelementen, die verdeeld kunnen worden in verschillende leeftijdsgroepen. Na iedere reactorcyclus (normaliter 12 à 18 maanden) worden de oudste elementen verwijderd, de resterende elementen herschikt, en nieuwe, ongebruikte elementen op de vrijkomende posities geplaatst. Een schema dat aangeeft waar welk element geplaatst moet worden, heet een herlaadpatroon. Het gekozen herlaadpatroon heeft veel invloed op de efficiëntie van de reactor. In de huidige praktijk worden herlaadpatronen vaak ontworpen met gebruik van technieken die in de wiskunde bekend staan als direkte zoekmethoden.

De aanpak van herlaadpatroonoptimalisatie in dit proefschrift is gebaseerd op methoden die gebruik maken van de afgeleiden van functies. Het gedrag van de reactor tijdens de cyclus wordt beschreven met een stelsel differentiaal vergelijkingen. Discretisatie van deze vergelijkingen geeft een stelsel algebraïsche vergelijkingen, die gebruikt kunnen worden in een wiskundig optimalisatiemodel. De doelstelling van ons onderzoek was om na te gaan wat de voordelen en nadelen van zo'n aanpak zijn, en om te onderzoeken of methoden die gebruik maken van afgeleiden nuttig kunnen zijn voor herlaadpatroonoptimalisatie. Hierbij was het de bedoeling om geschikte wiskundige modellen en algoritmen te zoeken of te ontwerpen.

Na een inleidend hoofdstuk volgt in hoofdstuk 2 een beschrijving van de onderliggende fysica, met de bijbehorende differentiaalvergelijkingen. Een zeer gedetailleerd model dat alle details zou bevatten van de kern, de koelvloeistof, drie ruimtelijke dimensies, en dergelijke, zou zo groot zijn dat het evalueren van één enkel herlaadpatroon al heel veel rekentijd zou vergen. Daarom hebben we een optimalisatiemodel ontworpen dat enerzijds de essentiële elementen bevat en een redelijke indicatie geeft van het rendement en de veiligheidsmarges van een patroon, terwijl het anderzijds klein genoeg is om gebruikt te worden in een optimalisatie. In het model is het mogelijk om fractionele waarden te gebruiken voor de variabelen die de toewijzing bepalen. Met andere woorden, brandstofelementen kunnen onderling worden gemengd. Dit is een voordeel voor de door ons geteste methoden. Natuurlijk dient de optimalisatiemethode wel een uiteindelijke oplossing te geven waarin de brandstofelementen geheel zijn. Het basis model is uitgebreid zodat ook de concentratie van 'burnable absorbers' kan worden geoptimaliseerd. Deze stoffen worden toegevoegd aan de splijtstof om het vermogen in nieuwe elementen tijdelijk te reduceren, wat leidt tot een beter gebalanceerde vermogensverdeling.

Hoofdstuk 3 bevat een uitgebreid overzicht van methoden die in bestaande literatuur worden beschreven voor herlaadpatroonoptimalisatie. De meeste artikelen beschrijven het gebruik van direkte zoekmethoden, zoals pairwise interchange, simulated annealing, en evolutionary algorithms. Enkele artikelen beschrijven het gebruik van methoden die gebaseerd zijn op functie-afgeleiden. De daar gebruikte modellen zijn gewoonlijk kleiner en minder gedetailleerd dan ons model. Verstoringstheorie is een algemeen hulpmiddel dat in beide soorten toepassingen wordt gebruikt. Dit geldt ook voor het gebruik van vuistregels die de zoekruimte moeten beperken.

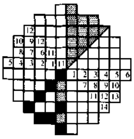
Het vierde hoofdstuk geeft een beschrijving van bruikbare gemengd-geheeltallige niet-lineaire optimalisatiemethoden. In de verschillende implementaties die door ons zijn gebruikt, worden varianten gebruikt van sequentieel quadratisch optimaliseren, van de 'augmented lagrangian trust region' methode, en van de '(generalized) reduced gradient' methode. Evenals de direkte zoekmethoden geven deze methoden lokaal optimale oplossingen voor niet-convexe problemen. Geheeltallige oplossingen worden verkregen via afrondingsheuristieken, via de branch-and-bound methode of met behulp van 'outer approximation'. Verder kunnen snedes worden toegevoegd om het vinden van geheeltallige oplossingen efficiënter te maken. In ons geval hebben we snedes gebruikt om slechte locale oplossingen weg te snijden. Een andere aanpak van het optimalisatieprobleem maakt gebruik van globale optimalisatietechnieken.

Hoofdstuk 5 vormt de kern van het proefschrift. Hierin worden alle zaken besproken die aan bod kwamen bij de implementatie. Wegens de niet-convexiteit en geheeltalligheid van het probleem waren vele specifieke aanpassingen nodig, en verschillende programma's moesten worden getest. Het vinden van een goed startpunt voor de optimalisatie was absoluut noodzakelijk. We hebben enkele startpuntgeneratoren ontwikkeld die bevredigende oplossingen geven. Andere belangrijke aspecten voor het vinden van goede oplossingen zijn: strakke grenzen op de variabelen, het toevoegen van hulp-beperkingen en het initieel relaxeren van bepaalde constraints. Andere –al of niet succesvolle– aanpassingen zijn eveneens onderzocht en worden besproken, waaronder het toevoegen van snedes. De keuze van de tijdsdiscretisatie was belangrijk in verband met stabiliteit. Een combinatie van gemengd-geheeltallige niet-lineaire optimalisatie met een pairwise interchange heuristiek is ook bekeken en getest. De resultaten kunnen worden gevisualiseerd met een grafische interface, die alle karakteristieken van een herlaadpatroon toont. Ook geeft deze interface het verloop van de pairwise interchange heuristiek weer.

De uiteindelijke testresultaten zijn beschreven in hoofdstuk 6. Deze zijn verkregen met een commercieel pakket dat gebruik maakt van 'outer approximation', en met een eveneens commerciële implementatie van de 'generalized reduced gradient' methode. Dit laatste pakket is geïntegreerd met een eigen afrondingsprocedure en met eigen pairwise interchange algoritmen. Deze combinatie geeft resultaten die kunnen wedijveren met die van direkte zoekmethoden, terwijl de rekentijden veel gunstiger zijn. Bovendien kan ons model vrij eenvoudig worden uitgebreid met continue variabelen, zoals bij het modelleren van (continu te variëren) burnable poisons. Zo'n uitbreiding is niet evident bij het gebruik van direkte zoekmethoden.

Een kort zevende hoofdstuk is tenslotte nog gewijd aan alternatieve methoden en varianten die kort zijn bekeken. Hieronder vallen andere algoritmen voor niet-lineaire optimalisatie, methoden voor globale optimalisatie, parallellisatie, en een semidefinite relaxatie.

Arie Quist, Mei 2000 TU Delft



Chapter 1

Introduction

Nuclear power plants are an important source of electricity in many countries. These power plants are usually fueled by uranium. Natural uranium is composed of different types of uranium (isotopes), and only 0.7 % is suited for fission in the reactor. For nuclear reactor operation, uranium is needed with a concentration of fissionable material of about 3 %. In order to obtain uranium with this percentage of fissionable material, an expensive enrichment procedure has to be applied. The uranium that is used in a power plant is therefore very costly, and it must be utilized efficiently. Furthermore, optimal use of the available uranium will reduce the nuclear waste. *Nuclear fuel management* is concerned with the question of how to utilize the available uranium in a safe and efficient way. For answering this question adequately, engineering knowledge only is not sufficient, but also advanced *mathematical optimization* methods are needed. Our primary goals are the development of suitable mathematical optimization models of nuclear fuel management problems and the design of an optimization system to solve those models.

A brief introductory description of the two disciplines –fuel management and mathematical optimization– is given in the next two sections. This elementary introduction enables us to formulate our fuel management optimization problem more precisely in Section 1.3. An outline of the thesis is presented in Section 1.4.

1.1 Nuclear reactor fuel management

A schematic picture showing the main components of a nuclear power plant is given in Figure 1.1. The heart of the plant is the nuclear reactor core, containing a number of vertically positioned rods with nuclear fuel (usually uranium). During normal reactor operation, nuclear fission reactions in the fuel cause heating of the liquid that flows through the core between the rods. Usually, this liquid is just water. Depending on the type of the reactor, the water may start boiling, or in order to prevent boiling it may be kept under very high pressure. Anyhow, the heated water is used to drive a turbine, in most cases via a heat exchanger and a secondary water circuit. The turbine in turn drives an electricity generator, and the obtained electricity is distributed via the electricity network.

We do not include the water circuits, turbines, *etc.* in our models, we concentrate only on the reactor core containing the fuel elements. A top-down view of a possible reactor core is given

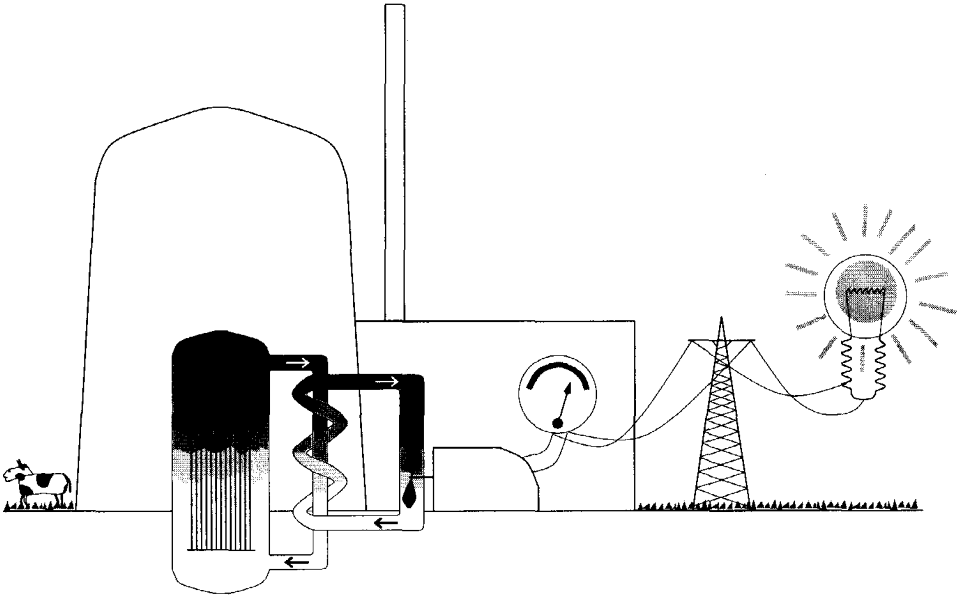


FIGURE 1.1: Schematic picture of a normal operating nuclear power plant.

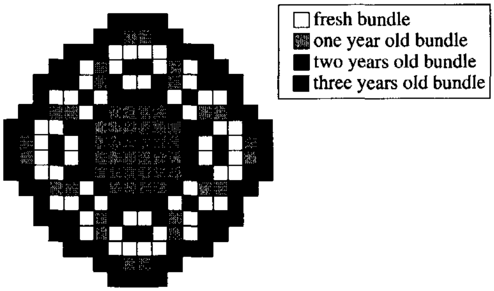


FIGURE 1.2: Schematic view of a fuel loading pattern.

in Figure 1.2. The core consists of a fixed number of (square) holes, the fuel nodes. Each node is filled with a fuel element. A fuel element, that may have a width of about 10 to 25 cm is a construction that contains a large number (say a hundred) of tiny cylindrical fuel pins or rods with a diameter of about 1 cm. (Figure 1.3). We are only interested in the element as a whole, and often use the term 'bundle' as an equivalent of 'element'. We consider an element as if it were a solid square rod of uniform material.

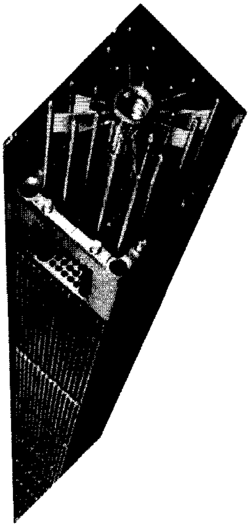


FIGURE 1.3: One fuel element, consisting of a number of fuel pins. The spider-like construction on the top of the element is used to move control rods in and out of the element.

In some of the nodes in the core, a number of rods may be control rods instead of fuel rods. Control rods contain only a certain absorbing material. Fuel rods emit neutrons that keep the chain reaction alive. The amount of neutrons that are supplied determines the activity of the core. If too many neutrons are emitted, they can be captured by control rods. Since control rods can be moved in and out of the core during reactor operation, they can be used to regulate the power level in the core. In most of our experimental models, the control rods are neglected. In real reactor core computations, one has to take them into account.

Not all fuel elements in the core have the same composition. One of the reasons is that during reactor operation, the power may be unevenly supplied over the core, and as a consequence one bundle looses its fuel more rapidly than another. Another reason is that, depending on the loading strategy, it may be advantageous even at the start of the reactor operation to fill the different nodes with bundles that have different fuel concentration. If, for example, all bundles in the core would have the same fuel composition, one could imagine that the center of the core would be much more heated than the outer parts of the core. This shows that the fuel has to be distributed over the core in a smart way.

From time to time, usually each year, the reactor operation is suspended for reloading and maintenance. Fuel bundles[1] may be discharged from the core, new bundles may be put into

---

the core, and the remaining bundles may be reshuffled. This operation is called the reloading operation, and the operational time between two reloading operations is called a cycle. In a usual reloading operation, only a part of the bundles is discharged from the core, and fuel bundles that still contain a reasonable amount of fuel are left in the core. In the new configuration, they are used to create the necessary diversification in the core. In order to obtain an efficient and safe core in the next cycle, optimal positions of all bundles have to be found.

The placement of the bundles in the core has a significant influence on the behavior of the reactor in the forthcoming cycle. This is illustrated in the Figures 1.4 and 1.5. The left parts of these figures give a top-down view of the core layout. This type of pictures is used frequently in the sequel of the thesis. The different shades of gray denote the different ages of the bundles. A white bundle is freshly inserted in the core. A light gray bundle has already been in the core during the previous cycle. A dark gray bundle has been in the core for two previous cycles, and a black bundle has already stayed in the core for three consecutive cycles. In the depicted figures, and actually in all models in this thesis, a strategy is used where exactly a quarter of the bundles is removed during each reloading operation, and where each bundle stays in the core for four consecutive cycles.

The figures give an indication of the importance of a good loading pattern, by showing cases where problems may arise. Figure 1.4 illustrates a situation where it may occur that the chain reaction cannot be maintained. Although the core may initially start up, the old bundles in the center are emitting too few neutrons to keep the chain reaction alive for the whole fuel cycle. The fresh, active, bundles loose a large fraction of the emitted neutrons in the surrounding water. As a result, the neutron density in the core decreases, the number of neutrons that cause a new fission is too low and the reactor stops operating. Figure 1.5 illustrates another case. In the outer parts of the core almost nothing is happening, but in the center there is a large fresh fuel concentration. Sufficient neutrons emitted in the core center cause a new fission reaction to cause a high temperature in the center. This has to be controlled by inserting neutron absorbing control rods in the center. But since the outer parts give almost no power, the total power then becomes too low to to maintain the required power level.

These two cases show two requirements for good loading patterns. The bundles should be arranged such that a sufficient power level can be obtained during the whole fuel cycle. Also, the power should be distributed evenly over the core, to prevent local heating. This leads to cores where too large concentrations of fresh or very old bundles are omitted. A core that might operate well is given in Figure 1.2 at page 2.

In nuclear reactor fuel management, one looks for such an acceptable loading pattern, and one tries to find a loading pattern that gives optimal performance in the forthcoming cycle, where 'performance' still has to be defined more precisely. In nuclear reactor operation, it is very important to find loading patterns that have optimal or nearly optimal yield. If by a better loading pattern the yield can be improved by only 1 %, this may already save half a million dollars per year for a medium-sized power plant.

Due to the complexity of the problem, mathematical models and methods have to be used

---

[1] We use the terms fuel bundle and fuel element interchangeably. In nuclear physics literature, the term fuel bundle is often used for one fuel pin in a fuel element.

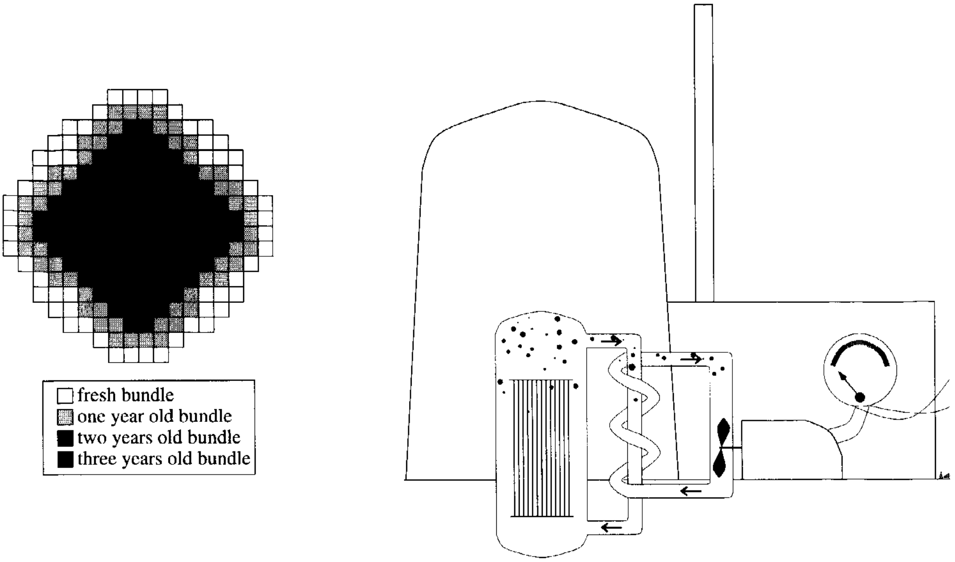


FIGURE 1.4: Loading pattern (schematic picture at left) that may become subcritical. In such a case, the chain reaction cannot be kept alive, the core dies out, and the electricity generator stops operating.

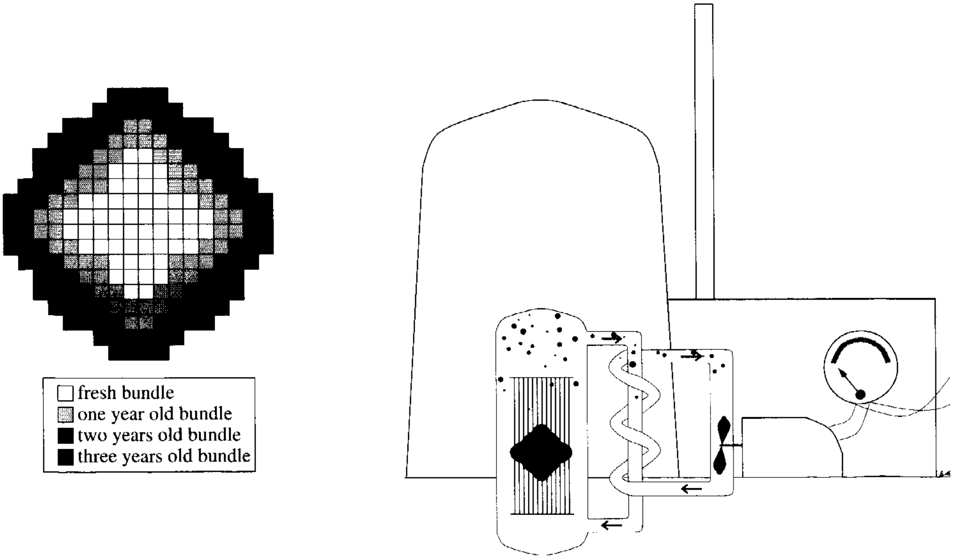


FIGURE 1.5: Core (left) with an overheating problem. Due to uneven power distribution over the core, the center would get overheated if the core was used at full operational level.

to find a loading pattern with optimal yield. Given the properties of the different fuel bundles during reloading, one may predict the behavior of the core for a specific loading pattern in the forthcoming cycle by solving a set of time-dependent partial differential equations. Thus, theoretically, it is possible to compute in advance the characteristics of the core for all possible loading patterns and to select the optimal one. In practice, this is not possible: for a core such as the one in Figure 1.2 there are $6 \cdot 10^{25}$ possible loading patterns, even if octant symmetry is assumed and bundles of identical age are considered equal, which is a very rough assumption. The fastest supercomputer nowadays (and in the visible future) would require billions of years to evaluate such a huge number of loading patterns.

We can conclude that it is not sufficient to have a tool that computes only the characteristics of a fixed loading pattern, but additional mathematical algorithms are required that are able to find optimal or close to optimal loading patterns, without evaluating all possible loading patterns. Such algorithms are developed in the discipline of mathematical optimization. Although we did not find a historical overview –loading pattern optimization is a quite young area–, there are literature references that go back to the late sixties and early seventies, where techniques from mathematical optimization are applied to find good loading patterns [56, 87, 113, and the references therein]. Due to the difficulty of the problem, however, no solution method that is known can guarantee that it finds the best solution, and there is still an ongoing research to find better methods and techniques for finding better loading patterns, and the current thesis fits in this area. Before giving a more precise description of the scope of our work, attention is paid in the next section to the area of mathematical optimization.

1.2 Mathematical optimization

This section starts with a short historical overview of the field, which is inspired by [24, 71]. This overview leads to the description of the linear optimization model. Extensions are discussed in Section 1.2.2. A small example application, illustrating the use of mathematical optimization for a simple problem that shares several properties with fuel management optimization, is shown in Section 1.2.3.

1.2.1 Historical overview and basics

The roots of mathematical optimization are quite old. The method of Newton (end of the 17th century) for solving systems of equations plays a fundamental role in many optimization algorithms. Duality, one of the fundamental concepts in mathematical optimization, was known at the end of the 18th century [68]. Lagrange (end of the 18th century, [70]) introduced a duality theory for nonlinear optimization that is until now extremely important. Other well-known names are Cauchy (steepest descent method, mid 19th century) and Farkas (extension of duality theory, end 19th century). All these people built the basic ingredients for optimization methods. Still, they did not use optimization methods for several reasons. They developed a theory of equalities, but did not make the extension to inequalities, which is crucial for mathematical optimization. Furthermore, they did not have the computer power to solve practical optimization problems. Thirdly, the need for mathematical optimization was simply not felt.

The real breakthrough of mathematical optimization, and the initiation of a dedicated discipline, called operations research, was due to enormous logistics problems of the allied forces in World War II. Huge numbers of people, clothes, weapons, cars, ships, airplanes, *etc.* and huge amounts of oil, food and the like had to be in time at the places where they were needed. In order to solve these problems, staff members with mathematical insight were appointed and they started to tackle the problems in a systematic way, by developing mathematical models and designing algorithms to solve these models. After the end of the war, they as well as other researchers continued developing those models and algorithms. Available resources were defined as parameters in the system. Decisions that had to be made were modeled as variables. When making the decisions a large set of constraints and requirements had to be respected. Finally, and this turned out to be a new concept at that time, an objective should be optimized that was defined as an explicit function of the decision variables. This objective could guide the search to obtain a solution that not only satisfies all requirements –if such a solution exists– but that is also optimal, *i.e.*, the best possible, with respect to a pre-specified goal. In 1947 this lead to the development of the linear simplex method by George Dantzig [23, 24]. Using this simplex method, it was possible to find an optimal solution to problems as just described, as long as the different relations, requirements, constraints and objective function could be defined as linear functions of the decision variables. It is guaranteed that such a solution is really (globally) optimal with respect to the defined objective function, which means that no better solution with respect to this objective function exists that still satisfies all restrictions.

Without computers, the practical use of the simplex method was limited to a small number of variables, since it would otherwise lead to a computational burden. During the second half of this century, due to the rise of modern computers, larger and larger problems could be solved. Nowadays, efficient implementations of the linear simplex method and the more recently developed interior point methods [110] solve linear optimization problems with hundreds of thousands of variables and constraints within acceptable time. Models of such sizes are extensively used in for example the airline industry, leading to savings of billions of dollars [8]. Many other linear models give drastical savings in various branches of industry.

1.2.2 More complicated models

Although many real-life problems can be solved by using linear optimization, there is also a bunch of optimization problems that cannot be modeled just by using linear relations. Here we mention two generalizations, that play an important role in the fuel management optimization problem.

The first is integrality of decision variables. An airline company cannot decide to sent half a plane to New York and the other half of the plane to London. In the same way, it is not possible to position half a fuel bundle in one position in a reactor core, and the other half of the bundle in another position. In Section 4.1 it is explained that this makes the optimization process much more complicated. Although methods are developed for this type of problems, the success of such methods is strongly dependent on the specific structure of the problem at hand. Clever specializations of the basic methods may be necessary. The algorithms for integer optimization basically fall apart into two main categories. On the one hand there are solution methods that are exact linear problems. These methods usually make use of linear or convex optimization algorithms in which the integrality requirement is ignored, embedded in an algorithm for finding integer solutions. The other category of algorithms are neighborhood search methods. Such methods 'jump' from one integer solution to another in a smart way, until they cannot find better neighboring solutions. These methods are not exact: they may get stuck at a non-optimal solution, but they are generally simpler and faster than exact algorithms.

The second generalization that makes optimization more complicated than linear optimization is nonlinearity in the relations between the decision variables or/and nonlinearity in the objective function. This may cause several more or less severe problems. One problem is numerical instability. Small changes in one variable may cause large changes in other variables. Another problem is nonconvexity, a feature (explained in Section 4.1) that makes it almost impossible to guarantee that a calculated locally optimal solution is really the best possible. In our problem, this makes it practically impossible to find the overall best loading pattern with respect to the given objective function, or if we find it by chance, one cannot make sure that it is the overall best solution. Nonconvexity may also disconnect the feasible area, which in our case means that it might be impossible to reach certain loading patterns, depending on the starting point of the algorithm. As with algorithms for integer problems, the efficiency of algorithms for nonlinear optimization problems strongly depends on the problem at hand. For current practical problems, it is generally possible to find a solution that is optimal in its neighborhood for problems with several thousands of variables and constraints. Guaranteeing that a solution of a nonconvex problem is really globally optimal, is in general only possible for problems with at most a few hundred variables by using global optimization algorithms.

The nuclear reactor loading pattern optimization problem contains both types of complications: as we shall see, it is nonlinear, even nonconvex, and also contains integral decision variables. This makes the problem very hard to solve. Algorithms based on both solution approaches for integer optimization may be applied. On the one hand, it may be possible to jump from pattern to pattern in a clever way, in order to arrive at some hopefully good pattern. This is how it is usually done. The other way is to apply some nonlinear optimization algorithm in order to find a local optimal solution, in which the integrality constraint is ignored. Such a tool should be embedded in an other algorithm that finds integer solutions. This is the direction that will be investigated in this thesis.

The nuclear reactor loading pattern optimization problem is an optimization problem where for a given set of optimization variables, the objective value is determined by the solution of a set of differential equations, where the optimization variables are parameters to the differential equations. In other words, the problem is to find optimal settings of the parameters of a system of differential equations. In the next section, we present a small example to illustrate the approach that we are using. More sample applications of differential equations in various mechanical systems can be found in [3, 125]; many of these examples can be extended to an optimization context, just as the example below.

1.2.3 Example application

Consider the following simple example, illustrated in Figure 1.6. We are given a small boat that has to cross a river, and to arrive at a landing stage at the other side. The water in the river

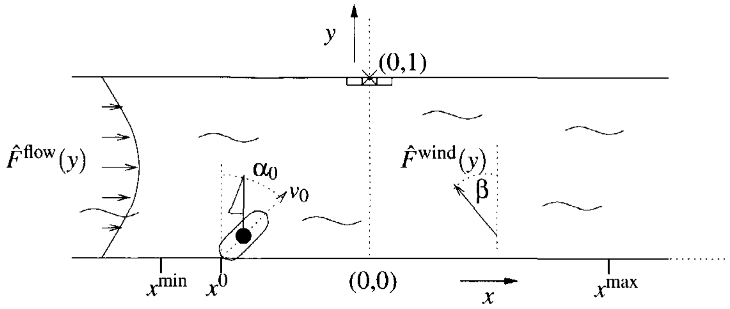


FIGURE 1.6: The boat has to cross the river.

flows at a position-dependent velocity as is indicated by the arrows on the left, giving a force $\hat{F}^{\text{flow}}(y) = \hat{F}_0^{\text{flow}} \cdot y(1-y)$ in x-direction to the boat. The boat (with mass 1) has to reach the point (0,1) at the other side at a time that is scaled to $t = 1$, and can start at $t = 0$ at any point $(x^0, 0)$ with x^0 in the range $[x^{\min}, x^{\max}]$, in any direction $\alpha^0 \in (-90°, 90°)$ and with any velocity $v^0 \in [0, v^{\max}]$. The trajectory traversed by the boot is expressed by the variables $(x(t), y(t))$. Besides flow, the boat also is faced with a resistance that is proportional to the square of its velocity with parameter ω. The boat cannot be steered, but the wind is blowing in a direction β, and a sail can be raised and lowered dynamically during the crossing of the river, to obtain a force between 0 and $\hat{F}^{\text{wind}}$ in direction β. The fraction of the sail that is raised at time t is to be determined as a value $s(t) \in [0;1]$.

We want to find an initial position x_0, an initial velocity v_0, an initial direction α_0, and a function $s(t)$ describing how much the sail should be raised at any time $t \in [0;1]$ such that we arrive at position (0,1) at $t = 1$, and such that the maximum deviation d in x-direction from $x = 0$ during the crossing is minimized. The x- and y-coordinates $x(t)$ and $y(t)$ of the boat during the crossing are determined from Newtons relation $F = m \cdot a$, that in our case translates into a system of differential equations for the x- and y-direction. The acceleration a is the second derivative of the position; the force is the sum of the flow force and the wind force, minus the resistance force:

$$\begin{aligned}
\frac{d^2x(t)}{dt^2} &= y(t)(1-y(t))\hat{F}_0^{\text{flow}} + s(t)\hat{F}^{\text{wind}}\sin\beta - \omega\left(\frac{dx(t)}{dt}\right)^2, \quad t \in (0,1), \\
\frac{d^2y(t)}{dt^2} &= 0 + s(t)\hat{F}^{\text{wind}}\cos\beta - \omega\left(\frac{dy(t)}{dt}\right)^2, \quad t \in (0,1).
\end{aligned} \tag{1.1}$$

Associated with the differential equations are the boundary conditions:

$$\begin{aligned}
\left(\frac{dx(t)}{dt}, \frac{dy(t)}{dt}\right)_{t=0} &= (v^0 \sin\alpha^0, v^0 \cos\alpha^0), \\
(x(0), y(0)) &= (x^0, 0), \\
(x(1), y(1)) &= (0, 1).
\end{aligned} \tag{1.2}$$

Here, v_0, α_0 and $s(t)$ are the control parameters. Outside an optimization context, the system of differential equations (1.1) with boundary conditions (1.2) can be solved to determine $x(t)$ and $y(t)$ for fixed values of x^0, v^0, α^0 and $s(t)$[2]. In an optimization context, the control parameters appear as variables in the problem. Furthermore, an additional variable d is introduced to measure the maximum distance from $x = 0$, that has to be minimized. This leads to the following optimization model:

$$
\begin{array}{rl}
\underset{x^0,v^0,\alpha^0,s(t),x(t),y(t),d}{\text{minimize}} & d \\
\text{subject to} & \text{Equation (1.1)}, \\
& \text{Equation (1.2)}, \\
& d \geq (x(t))^2, \quad t \in [0,1], \\
& x^0 \in [x^{\min}, x^{\max}], \\
& \alpha^0 \in (-90^\circ, 90^\circ), \\
& v^0 \in [0, v^{\max}], \\
& s(t) \in [0;1].
\end{array}
\tag{1.3}
$$

By choosing a suitable discretization of the time-axis, one may write (1.3) as a nonlinear optimization model with a finite number of variables and constraints, that can be solved using standard nonlinear optimization techniques. When solving such a model, one implicitly finds both an optimal parameter setting and the corresponding solution of the differential equations in one solution process.

In principle, any equation that can be stated as an algebraic equation can be handled by a nonlinear optimization model. It makes no difference whether the equation is a differential equation, a boundary constraint, a range constraint for a variable, or another limitation imposed to the solution. This idea is also used in our solution approach of the nuclear fuel management optimization problem. We incorporate the differential equations, the safety limitations and the reloading operations all together as sets of algebraic equations in our optimization model. An extension which is not present in the river-crossing example, is the presence of the integrality constraints. Speed and velocity are quantities that may vary continuously; the placement of bundles, however, is a discrete operation: it is not allowed to spread one fuel element over different positions. This makes the fuel management optimization problem non-standard and more complicated, but the basic ideas of using nonlinear optimization are the same as in the river-crossing example.

---

[2] For simplicity of presentation, we assumed here that the parameters are chosen such that a solution exists, which is not necessary the case since the number of boundary conditions is larger than the number of degrees of freedom. Alternatively, one might let v^0 and α^0 vary to reduce the number of boundary conditions to four.

1.3 Motivation, aims and scope

Prior to our research, a study is performed by de Klerk *et al.* [65], in which a very simplified model of the reload pattern optimization problem is implemented in standard nonlinear mixed-integer optimization algorithms. This study indicated that nonlinear optimization might be a good candidate approach for the reload pattern optimization problem, and it forms the motivation for the current study.

In nuclear reactor fuel management optimization, one is concerned with designing loading patterns for nuclear power plants, such that some objective function related to the yield of the power plant in forthcoming fuel cycles is maximized. Currently used optimization methods may be very different, but share the property that the objective function is evaluated at 'integer' loading patterns only. Derivative information from the equations that describe the physical state is not used. An exception to this is the use of perturbation theory (Section 3.3.1), but this is only useful for selecting integer patterns that are very close to a given pattern. In contrast, derivative-based nonlinear optimization algorithms make extensive use of derivative information with respect to the change in any variable. During the optimization, these algorithms may place fractions of bundles. In one iteration it may for example replace one eighth of three bundles, and together three quarters of five other bundles. Special care is needed so that in the end an integer loading pattern will be found. This approach opens new ways for finding optimal solutions. The purpose of our research is

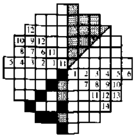
> *to explore the usefulness, to find the advantages and disadvantages, and to evaluate whether derivative based methods are potentially useful in loading pattern optimization, by developing suitable mathematical models and algorithms.*

A first step to reach this goal is the development of appropriate optimization models. Different classes of optimization methods require different model variants. Suitable derivative based optimization tools have to be found. In order to make a fair comparison with existing techniques, available state of the art derivative-based optimization algorithms have to be tailored to the newly developed models. Search heuristics that are currently in use must be explored and tested in order to be able to compare derivative based methods to current practice.

Criteria that are used to compare the algorithms are flexibility, robustness, solution quality and computational resource usage of our methods, compared to other fuel management optimization methods.

1.4 Outline

This thesis is organized as follows. Chapter 2 contains an introduction to reactor physics, in which the mathematical equations are derived that describe the evolution of the reactor core during an operating cycle. These equations are combined with equations describing fuel reloading operations and with further operational constraints. Further, an objective function for the optimization is defined, finally resulting in a basic optimization model that will be used in the rest of the thesis. Some variants and extensions of this model are discussed as well. In Chapter 3 an overview is presented of solution methods from the reactor physics literature, that are potentially usable to solve the reload pattern optimization problem. Advantages and restrictions of the different methods are briefly discussed. Then, Chapter 4 focuses on mixed-integer nonlinear optimization techniques that are used in our approach. The mathematical description of these techniques is rather general. Our problem specific additions and modifications to these methods are discussed in Chapter 5. This chapter gives a rather detailed overview of the different parts of the algorithms that we developed, and it is completed with an overview of our actual software implementations. Computational results obtained with these implementations are given and discussed in Chapter 6. During the research we also considered various solution methods that went beyond the main scope of the thesis. Some of them were interesting enough to be mentioned in Chapter 7. Finally, conclusions and possible future research directions are discussed in Chapter 8.



Chapter 2

Problem statement

In this chapter we give a description of the physics underlying the reload pattern optimization problem, which is then used to derive mathematical formulations of this problem.

In the reload pattern optimization problem, a number of fuel bundles are given, which have to be placed into positions (nodes) in a reactor core. The bundles will stay in the nodes for one reactor operation cycle, which is usually a year, and we want to compute in advance the behavior of the core when the core is utilized to obtain a given amount of power during the cycle. Using such a prediction, we can search for the assignment of bundles to positions (the *loading pattern*) that is optimal with respect to a pre-specified goal, or objective function, which is related to the 'yield' of the reactor in the forthcoming cycle.

In order to find such an optimal loading pattern, the behavior of the core has to be described by a mathematical optimization model. This model will consist of a number of variables, the values of which can be varied in order to find the best loading pattern. The restrictions on these variables, and the relations between them, are described using a set of equations and inequalities known as constraints. A number of fixed parameters such as the core geometry, required power level, *etc.* are known as well. The value that we want to optimize is described in terms of the variables and parameters and is known as the objective function.

In the mathematical models, two types of variables can be distinguished. The *decision* variables describe the choice of the actual loading scheme, defining where the specific bundles will be placed, and additionally they describe what properties the bundles have at the beginning of a cycle (BoC). The *dependent* variables describe the physical properties such as the neutron flux and the 'yield' (to be defined later) in the reactor core in the forthcoming cycle, when a valid assignment of the decision variables is given.

The constraints can also be divided into different types. There are constraints that restrict the possible values of the decision variables. These include for example the assignment constraints, describing that each bundle is placed in exactly one node, and each node contains exactly one bundle.

Given an assignment of the decision variables, another set of constraints implicitly defines the values of the dependent variables. These constraints are derived from the differential equations that describe how the neutron flux changes in time and how the amount of fuel decreases. In physical terms, it may sound a bit strange to denote these descriptions as constraints. In a mathematical optimization context however, they are just seen as restrictions on the possible values of the dependent variables.

A third set of constraints adds further restrictions on the possible values of the dependent variables. These include safety constraints that forbid certain values of some physical variables, *e.g.*, in order to prevent a too large heat generation in some parts of the core. Since the values of the dependent variables depend on the values of the decision variables, these constraints also indirectly limit the possible values of the decision variables.

The parameters in the problem describe the layout of the core, the possible properties of fresh fuel bundles, required power level, cycle length and a number of other things. In fact, the choice which values are parameters and which are variables in the model is not unique. Depending on a specific situation, one may for example fix all properties of fresh bundles, or let some properties (*e.g.*, the concentration of some particles in the fuel) become decision variables. As another example, the cycle length may be fixed in advance, or it may be left as a decision variable.

Based on the variables and parameters, an objective function is defined. The objective has to make a clear distinction between good and bad loading patterns. There is some freedom in how to choose this function. Several different objective functions exist in literature, and an objective function has to be found that is suited for our purposes and fits in our model.

We also have to choose the level of detail of our models. In the next section, we start with a rather detailed model describing reactor physics in a general way. Since this description is far too detailed to be of practical use, it is simplified in a number of steps, and finally a model is derived to compute the dependent variables for a given initial reactor configuration. Some variants and possible extensions to this basic model are discussed. In Section 2.2 the decision variables are defined. The relations between the decision variables and dependent variables are described, this is the so-called reloading operation. Further, different elements that are needed to complete the optimization model are discussed in more detail. The third section gives a short overview of the different model variants that are used in this thesis.

2.1 Reactor Physics

This section starts with an introductory description of the neutron fission process. Starting from a detailed model in terms of the neutron transport equation, a series of simplifications lead to a formulation in terms of a diffusion equation, that can be handled much better in practical implementations. However, it can be further simplified by using a mathematical tool known as Green's function theory. Using spatial discretization, this finally leads to a basic stationary description of the neutron flux in the core at the end of Section 2.1.4. Time-dependence is studied in Section 2.1.5, which completes the development of the basic physical model. It still lacks several details that are needed for real-life computations. Implementations of two important extensions, the inclusion of burnable poisons and the treatment of multiple types of nuclides individually, are worked out in two separate subsections. Some further extensions are discussed, without going into the details, in the last subsection. The derivation of the physical model is mainly based on the books [18], [28], [54] and [118]. The first book [18] handles more generally about the whole fuel cycling, starting from Uranium mining up to waste management, while the latter three handle more in depth about the underlying physics.

Before continuing, some definitions and notation are introduced. The term *nucleus* refers to the nucleus of a general atom. This nucleus is the 'center' of the atom, and is built of a number of *protons* having a positive charge of about 1eV, and a number of *neutrons* without charge. The various species of nuclei with different numbers of protons and neutrons are called *nuclides*. The number of protons in a certain nuclide determines its *atomic number*, representing its name and place in the periodic system. The total number of protons and neutrons of a nuclide is its *mass number*. Different nuclides with the same atomic number but with a different mass number are called *isotopes*. A specific nuclide is denoted by the notation ^A_ZX, where A is the mass number, Z is the atomic number, and X is the abbreviated name in the periodic system. Since the name in the periodic system is uniquely coupled to the atomic number, the latter number is usually omitted. For example, the Uranium isotope with mass number 235 is written as $^{235}_{92}$U, or just as ^{235}U.

2.1.1 Fission chain reactions

In a nuclear power plant, energy is obtained by splitting heavy nuclei into smaller nuclei. The reason why this is possible is that different nuclei have different binding energy per nucleon. *Binding energy* can theoretically be viewed as the energy that would be released per nucleon when a nucleus is being built up from separate nucleons. It arises from the fact that the mass of the nucleus is slightly less than the sum of masses of 'isolated' protons and neutrons (Einsteins famous formula $E = mc^2$ is at work here). Since the binding energy is highest for medium sized nuclei, transformation of a big nucleus into two medium sized nuclei delivers a large amount of energy (see Figure 2.1).

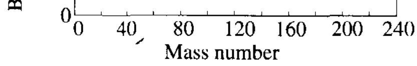


FIGURE 2.1: Binding energy as function of the mass number.

Before a fission can occur, an energy 'barrier' has to be passed: some activation energy has to be added to the nucleus. For nuclei with mass number smaller than 230, this energy barrier is so large that fission does not occur in nature. On the other hand, for very large nuclei with mass number greater than 260, this barrier has disappeared, so that these nuclei fall into parts spontaneously. The best nuclei for practical use in nuclear reactors have a mass number of around 235–240.

In a nuclear reactor core, the activation energy is supplied by the interaction of a nucleus with a free neutron. If this neutron is absorbed by the nucleus, the binding energy of the neutron, plus the kinetic energy of the neutron, may initiate a fission, leading to the emission of one or more new free neutrons. These free neutrons may again cause a fission reaction. In this way a chain reaction is obtained.

If all neutrons would cause another fission, the chain reaction would multiply very rapidly, resulting in an atomic bomb. However, neutrons may be absorbed by other nuclei, or they may get lost in the water surrounding the core. Some types of nuclei are particularly well suited for absorbing free neutrons, and they are used within control rods or in other mechanisms that regulate the reactor power. One of the important aspects of reactor operation is to find such loading patterns and such control schemes for the regulating mechanisms, that on the one hand excess neutrons can be captured effectively, while on the other hand there are enough neutrons in each part of the core to maintain the chain reaction.

The reactions that neutrons can undergo in the core and that are relevant for our work are mainly divided into three groups.

- Fission: the neutron is captured by a nucleus and causes fission of the nucleus into smaller nuclei, usually leading to the emission of one or more new neutrons.
- Caption: the neutron is captured by a nucleus, increasing the mass number of the nucleus by one.
- Scattering: the neutron collides with a nucleus, and is either scattered immediately, or it is absorbed, followed by emission of itself or another neutron, at a different energy level.

The question which reaction occurs in a particular case depends on the type of nucleus involved, the velocity of the neutron and the nucleus, and quantum-mechanical aspects that go far beyond the scope of this thesis. The number of nuclei and neutrons in the core is so large, (10^{22} nuclei and typically 10^8 neutrons per cm^3), that we can work in a statistical sense with probabilistic rates at which the different reactions will occur.

The probability that a certain interaction of neutrons with a certain type of nucleus occurs, is described by so called cross sections. The *microscopic cross section,* denoted by σ, can best be explained when we think of a nucleus as a solid ball. The microscopic cross section is then the cross section area of this ball with a plane through its center. In reality, the nucleus is not a solid ball, and the cross section only makes sense as a statistical number. It depends not only on the type of nucleus, but also on the type of interaction that is considered, and of the kinetic energy of a passing neutron. For neutrons at some energy levels, the *absorption* cross section area may be larger, while for neutrons with an other energy level, the *fission* cross section is the largest.

The microscopic cross section as function of the neutron energy is a general property of a nuclide type, and it is independent of the actual nuclide density in a particular situation. Once an actual *nuclide density* N for some nuclide is given, we can define the *macroscopic cross* *section* Σ, being the product of nuclide density and microscopic cross section:

$$\Sigma = \sigma N.$$

This cross section can be interpreted as the probability per unit path length of a free neutron to undergo the specified type of interaction with a nuclide of the specified type.

We will use the following subscripts to distinguish between different types of cross sections, related to the different types of reactions described:

- $\sigma_\mathrm{f}, \Sigma_\mathrm{f}$: Fission cross sections.
- $\sigma_\mathrm{a}, \Sigma_\mathrm{a}$: Absorption cross sections. This includes the absorption of neutrons due to fission, but excludes absorption followed by re-emission (scattering).
- $\sigma_\mathrm{s}, \Sigma_\mathrm{s}$: Scattering cross sections.

Cross sections can be defined for individual nuclide types, for groups of nuclides, or as the sum of cross sections for all nuclides. When we use a cross section only for one specified nuclide or group of nuclides, this will be clear from the context or it is denoted by superscripts, otherwise the cross section is the sum of cross sections of all nuclides. In the same way, position, time and neutron energy dependence either are denoted explicitly, or the cross sections are assumed to be average values[1]. These conventions with regard to notation also hold for the nuclide and neutron densities, as well as for the flux (which will be introduced in the next section).

2.1.2 Neutron transport

All relevant reactions in a reactor core are governed by the behavior of neutrons in the core. Neutrons emerge from fission or from external sources, move through the core at certain velocities and in certain directions, and either cause reactions in the core or get lost in the surrounding water. Therefore, we want to know the neutron flux at each time at each point in the core. To this end, we define the *angular neutron density*

$n(r, E, \hat{\Omega}, t)\, d^3r\, dE\, d\hat{\Omega} =$ The expected number of neutrons in a volume element d^3r around position r, with energy in the range dE around E, moving in a direction within the solid angle $d\hat{\Omega}$ around $\hat{\Omega}$ at time t.

The angular neutron density in the core is described by a neutron balance equation. This equation describes the rate of change of the angular neutron density for a given r, E and $\hat{\Omega}$ and in the ranges d^3r, dE and $d\hat{\Omega}$ at a given time t as follows:

$$\text{Rate of change} + \text{Loss} = \text{Gain}.$$

---

[1] In order to avoid any confusion of the macroscopic cross section sign Σ with the summation sign $\sum$ we consequently place indices straight under and above each summation sign, and sub- and superscripts after macroscopic cross sections.

Possible gain mechanisms are

1. Neutron sources in the volume element d^3r (*e.g.*, fission reactions).
2. Neutrons entering through the surface of the volume element d^3r.
3. Neutrons of arbitrary energy level and/or moving direction, that due to a scattering collision within d^3r change to an energy level within range dE around E and a moving direction within the solid angle $d\hat{\Omega}$ around $\hat{\Omega}$.

Possible loss mechanisms are

1. Neutrons leaking out through the surface of the volume element d^3r.
2. Neutrons within d^3r that suffer a collision, either absorption (which includes fission), or scattering.

Without describing the different mechanisms in detail, we give the resulting integro-differential equation ([28, p.113]; $n(r,E,\hat{\Omega},t)$ is abbreviated to n):

$$\begin{aligned}&\frac{\partial n}{\partial t} + v\hat{\Omega}\cdot\nabla n + v\Sigma_c n = \\ &\quad \int_{4\pi} d\hat{\Omega}' \int_0^\infty dE' v'\Sigma_s(E' \to E, \hat{\Omega}' \to \hat{\Omega}) n(r,E',\hat{\Omega}',t) + s(r,E,\hat{\Omega},t),\end{aligned} \tag{2.1}$$

where

- v is the neutron velocity;
- Σ_c is the total macroscopic cross section for any neutron capture (fission, absorption or scattering)
- $\Sigma_s(E' \to E, \hat{\Omega}' \to \hat{\Omega})\, d\hat{\Omega} dE$ is the scattering cross section for scattering from energy level E' and angle $\hat{\Omega}'$ to energy level E and angle $\hat{\Omega}$;
- $s(r,E,\hat{\Omega},t)$ is the neutron source term.

Equation (2.1) is known as the *neutron transport equation*. It needs an initial condition, describing the initial angular density for all positions, energies and directions. It further needs some boundary condition. A possible boundary condition can be that there is no neutron flow into the system at a boundary sufficiently away from the reactor core.

Instead of describing the neutron transport equation in terms of angular neutron density, it is beneficial to rewrite it in terms of the *angular neutron flux* ϕ, which is related to the angular neutron density in the following way:

$$\phi(r,E,\hat{\Omega},t) = vn(r,E,\hat{\Omega},t),$$

where v is the velocity of neutrons of the given energy level. The neutron transport equation then reads:

$$\begin{aligned}&\frac{1}{v}\frac{\partial\phi}{\partial t} + \hat{\Omega}\cdot\nabla\phi + \Sigma_c\phi = \\ &\int_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \Sigma_s(E' \to E, \hat{\Omega}' \to \hat{\Omega})\phi(r, E', \hat{\Omega}', t) + s(r, E, \hat{\Omega}, t).\end{aligned} \tag{2.2}$$

The neutron source term $s(r,E,\hat{\Omega},t)$ is not yet specified. The most important neutron source, and the only one considered here, is the neutron source due to fission, s_f:

$$s_f(r,E,\hat{\Omega},t) = \frac{\chi(E)}{4\pi}\int_{4\pi} d\hat{\Omega}' \int_0^\infty dE' \left[\nu(E')\Sigma_f(E')\phi(r,E',\hat{\Omega}',t)\right],$$

where

$\nu(E')$ is the average number of fission neutrons produced by a fission, induced by a neutron of energy E';

$\chi(E)$ is the energy distribution of fission neutrons.

The neutron transport equation may theoretically be discretized and used in an optimization model. However, even calculating one cycle for a fixed core configuration would be computationally very expensive, since a typical discretization would lead to a system of about 10^8 equations per time step ([28], p. 123). For this reason complexity reductions are absolutely necessary. One of the most useful and generally accepted simplifications is elimination of the angular variable vector $\hat{\Omega}$ by integration over all directions. Another simplification is to replace the energy dependence that can vary in a continuous range by a fixed, and not too large, number of 'average' energy levels. Other simplifications include separation of time- and space dependence, and in a later section, the assumption of uniform spatial distribution in larger parts of the core.

2.1.3 One and a half group diffusion theory

In order to derive a simplified model, (2.2) is integrated over $\hat{\Omega}$. This removes $\hat{\Omega}$ from all terms except for the term $\hat{\Omega}\cdot\nabla\phi$. By additional simplifications such as a truncated series expansion, which are not discussed in detail here, one can get rid of this term, resulting in angle-independent diffusion coefficients $D(r,E,t)$, and the corresponding diffusion equation[2]:

$$\begin{aligned}\frac{1}{v}\frac{\partial\phi(r,E,t)}{\partial t} &- \nabla\cdot D(r,E,t)\nabla_r\phi(r,E,t) + \Sigma_c(r,E,t)\phi(r,E,t) \\ &= \int_0^\infty dE'\Sigma_s(E' \to E)\phi(r,E',t) + \\ &\quad \chi(E)\int_0^\infty dE'\nu(E')\Sigma_f(E')\phi(r,E',t).\end{aligned} \tag{2.3}$$

---

[2] The dimension of the diffusion coefficient is [cm] instead of the more usual [$\mathrm{cm}^2\mathrm{s}^{-1}$], since the diffusion equation is defined for the flux ϕ instead of the neutron density n.

Although this equation already looks much nicer than the neutron transport equation (2.2), the energy dependence still causes huge complexity of the model. Therefore the energy range is divided into a number of energy groups, and (2.3) is integrated over the range of each group. By defining group-average variables and parameters, as well as scattering cross sections from one group to another in a proper way, a set of coupled diffusion equations, called the *multigroup diffusion equations* are obtained:

$$\begin{aligned}\frac{1}{v^{(g)}}\frac{\partial\phi^{(g)}(r,t)}{\partial t} &- \nabla\cdot D^{(g)}(r,t)\nabla\phi^{(g)}(r,t) + \Sigma_{\mathrm{t}}^{(g)}(r,t)\,\phi^{(g)}(r,t) \\ &= \sum_{g'=1}^{\bar{g}} \Sigma_{\mathrm{s}}^{(g',g)}(r,t)\phi^{(g)}(r,t) + \chi^{(g)}\sum_{g'=1}^{\bar{g}} \nu^{(g')}\Sigma_{\mathrm{f}}^{(g')}(r,t)\phi^{(g')}(r,t), \\ &\qquad g = 1,\cdots,\bar{g},\end{aligned} \tag{2.4}$$

where $\bar{g}$ is the number of energy groups, and the superscript $^{(g)}$ denotes a properly defined group averaged quantity. The notation $\Sigma_{\mathrm{s}}^{(g,g')}$ describes the scattering cross section from group g' to g.

In our final model, thermal reactors are considered in which a moderator slows down most neutrons to thermal energy. In this case, a rather accurate model can be obtained with only two energy groups: a *fast* and *thermal group*. Fast neutrons are characterized by high energy, while thermal neutrons have low energy. The boundary between the two is chosen in such a way that essentially all neutrons are born in the fast group ($\chi^{\mathrm{thermal}} = 0$), upscattering from the thermal to the fast group may be ignored, and diffusion in the thermal group is negligible. Denoting the fast group with superscript $^{(1)}$ and the thermal group with superscript $^{(2)}$, this leads to the following system of equations.

$$\begin{aligned}\frac{1}{v^{(1)}}\frac{\partial\phi^{(1)}(r,t)}{\partial t} &- \nabla\cdot D^{(1)}(r,t)\nabla\phi^{(1)}(r,t) + [\Sigma_{\mathrm{a}}^{(1)}(r,t) + \Sigma_{\mathrm{s}}^{(1,2)}(r,t)]\,\phi^{(1)}(r,t) \\ &= \nu^{(1)}\Sigma_{\mathrm{f}}^{(1)}(r,t)\phi^{(1)}(r,t) + \nu^{(2)}\Sigma_{\mathrm{f}}^{(2)}(r,t)\phi^{(2)}(r,t), \\ \frac{1}{v^{(2)}}\frac{\partial\phi^{(2)}(r,t)}{\partial t} &+ \Sigma_{\mathrm{a}}^{(2)}(r,t)\,\phi^{(2)}(r,t) = \Sigma_{\mathrm{s}}^{(1,2)}(r,t)\phi^{(1)}(r,t).\end{aligned} \tag{2.5}$$

This coupled system of equations is simplified by assuming that the shape of the flux does not change within the considered short-term time-intervals: that is, we assume that the flux $\phi(r,t)$ can be separated as $T(t)\phi(r)$. Substitution in (2.5) leads to a set of time-dependent equations, and to a set of space-dependent equations. In the short-term analysis, the time-dependent equations are ignored and the time-dependent variables are eliminated. This introduces an inconsistency in the space-dependent equations, that can only be resolved by introducing an additional degree of freedom to the system, *i.e.*, by introducing an eigenvalue to the system. The eigenvalue gives a way to indicate how the system changes with time, without actually needing a time-dependent variable. It is denoted as $\frac{1}{k^{\mathrm{eff}}}$, the inverse of the *effective multiplication factor*, which can be interpreted as follows:

$$k^{\mathrm{eff}} = \frac{\text{total neutron production rate}}{\text{total neutron loss rate}}.$$

The complete time-independent system is given below.

$$\begin{aligned}
-\nabla\cdot D^{(1)}(r)\nabla\phi^{(1)}(r) + [\Sigma_a^{(1)}(r) + \Sigma_s^{(1,2)}(r)]\,\phi^{(1)}(r) & \\
= \frac{1}{k^{\text{eff}}}\left[\nu^{(1)}\Sigma_f^{(1)}(r)\phi^{(1)}(r) + \nu^{(2)}\Sigma_f^{(2)}(r)\phi^{(2)}(r)\right], & \\
\Sigma_a^{(2)}(r)\,\phi^{(2)}(r) = \Sigma_s^{(1,2)}(r)\phi^{(1)}(r). &
\end{aligned} \tag{2.6}$$

The second equation of (2.6) can be substituted directly in the first equation, which eliminates the thermal flux $\phi^{(2)}$ from the system. Essentially this means that we have reduced the 2-group equations to a 1-group equation. This resulting equation is known as the $1^1/_2$ group diffusion equation. In order to simplify notation, we will remove the $^{(1)}$-superscript from the fast group flux.

This leads us to the final $1^1/_2$ group diffusion equation:

$$-\nabla_r\cdot D^{(1)}(r)\nabla_r\phi(r) + (\Sigma_a^{(1)}(r) + \Sigma_s^{(1,2)}(r))\phi(r) = \frac{1}{k^{\text{eff}}}\left[\nu^{(1)}\Sigma_f^{(1)}(r) + \nu^{(2)}\Sigma_f^{(2)}(r)\frac{\Sigma_s^{(1,2)}(r)}{\Sigma_a^{(2)}(r)}\right]\phi(r). \tag{2.7}$$

This is the basic equation used to derive the models used in this thesis.

The flux $\phi(r)$ acts as an eigenfunction in the system. This has two important consequences. Firstly, its magnitude is not yet defined, it describes only the shape of the flux. The magnitude will be fixed by appropriate boundary conditions, that will be described later on.

Secondly, the system (2.7) has infinitely many eigenfunctions, and thus infinitely many solutions. This is not a problem, however. It can be proven that all eigenfunctions are orthogonal. For a one-dimensional core, (2.7) is a Sturm-Liouville equation, which has the property that the eigenfunction corresponding to the largest value of k^{eff} is nonnegative over the whole space. Due to the orthogonality of the eigenfunctions, this implies that the other eigenfunctions have to be negative somewhere. For the more-dimensional case, the same property holds, but is much harder to prove [11]. A sketch of this proof is given in Appendix B.

We may conclude that the only solution of interest to us is the largest eigenvalue, the corresponding eigenfunction $\phi(r)$ which is nonnegative over the whole solution space.

Further simplifications

The eigenvalue differential equation (2.7) has to be simplified further in order to derive a nodal model. First, it is assumed that the diffusion coefficient $D^{(1)}$ is position independent. The fission cross section Σ_f strongly depends on burnup and is highly position dependent. The absorption cross sections, especially for the thermal group, also depend on burnup, but also on other properties such as the amount of fission products. Moreover, since the downscattering of neutrons is the most important mechanism of removing neutrons in the $1^1/_2$ group approximation, the removal cross section, which consist of the fast absorption cross section plus the downscattering cross section, is considered to be space independent. The downscattering cross section is approximately independent of burnup since it is mainly determined by the reactor coolant.

The remaining space dependent cross sections $\Sigma_f^1(r)$ and $\Sigma_f^2(r)$ are used to define the new space dependent variable $k^\infty(r)$ in the following way:

$$k^\infty(r) = \frac{\nu^{(1)}\Sigma_f^{(1)}(r) + \nu^{(2)}\Sigma_f^{(2)}(r)\frac{\Sigma_s^{(1,2)}}{\Sigma_a^{(2)}}}{\Sigma_a^{(1)} + \Sigma_s^{(1,2)}} . \tag{2.8}$$

The variable k^∞, the *infinite multiplication factor*, is interpreted as the ratio of local neutron production and absorption. Using this, the eigenvalue differential equation (2.7) can be transformed into

$$-L_f^2\nabla_r^2\phi(r) + \phi(r) = \frac{1}{k^{\text{eff}}}k^\infty(r)\phi(r), \tag{2.9}$$

where

$$L_f = \sqrt{\frac{D^{(1)}}{\Sigma_a^{(1)} + \Sigma_s^{(1,2)}}}, \tag{2.10}$$

is known as the *fast diffusion length.*

Boundary values

The eigenvalue differential equation (2.9) requires boundary conditions. The simplest boundary condition is defined on the outer region of the core, where it is required that[3]

$$\phi(r) = 0, \quad r \text{ on a boundary around the core.} \tag{2.11}$$

Sometimes symmetry of the core is exploited and only a quarter or an octant of the core is considered. In that case, the derivative of the flux should vanish at symmetry edges:

$$\frac{\partial\phi(r)}{\partial r} = 0, \quad r \text{ on a symmetry-edge.} \tag{2.12}$$

With these boundary conditions, $\phi(r)$ is still not fully determined, since it can be multiplied by a constant. This freedom is needed to fix the total power level in the core. Given

μ_f : the amount of energy released per fission

and the fission rate being the product of the fission cross sections and the flux, the local power density at position r is given by

$$\left(\Sigma_f^{(1)}(r) + \Sigma_f^{(2)}(r)\frac{\Sigma_s^{(1,2)}}{\Sigma_a^{(2)}}\right)\phi(r)\mu_f.$$

This quantity integrated over the whole core should be equal to the required total power level P_c. In order to write this constraint in terms of k^∞ instead of Σ_f, the two group average values $\nu^{(1)}$

---

[3] Neutrons can flow in the water surrounding the core, where they are absorbed. The boundary mentioned in this condition is therefore an artificial boundary at some distance of the core. In some cases, the core is surrounded by reflective material, leading to more advanced boundary conditions.

and $\nu^{(2)}$ in (2.8) are replaced by one total, properly averaged factor ν, which introduces only a very small error since the fast fission cross section is small compared to the thermal fission cross section. Let X denote the whole core region, then the power level fixing constraint is given by

$$\int_X \phi(r)k^\infty(r)\,dr = \frac{\nu P_c}{\mu_f(\Sigma_a^{(1)} + \Sigma_s^{(1,2)})}. \tag{2.13}$$

The diffusion equation with the corresponding boundary conditions cannot be solved analytically. Numerical methods have to be used to solve the problem. One possibility is the use of eigenfunction expansion methods. This is not a very practical method, since it is highly complicated by the fact that the core is non-homogeneous.

A better way is to use a direct discretization leading to a system of difference equations. This method is used in many core evaluation codes, but requires a rather fine discretization in order to obtain qualitatively good results. In the context of this paper, the model has to be used to find an optimal loading pattern by optimization of one large algebraic equation system, that includes the discretized diffusion equation. Since not only the solution, but also the parameters of the diffusion equation depend on the loading pattern, both the parameters and the solution of the diffusion equation are variables in this algebraic system. This will lead to an optimization model with a very large number of variables and many nonlinear and nonconvex constraints, that is rather difficult to solve. Such a detailed description is not necessary within an optimization context, since we can identify good loading patterns with a much coarser model (see also Section 2.1.8).

For the rest of this thesis, we restrict ourselves to a discretization with only one grid point per node. A direct discretization in this way would be much too coarse. A better result can be reached by using the theory of Green's functions. Using the latter, we can in a certain sense 'invert' the diffusion equation, which (when used carefully) allows the use of a much coarser grid, still with a reasonable solution quality. The resulting model is presented in the next section.

2.1.4 Green function model

The model as derived here is based on the theory of Green functions [28, 48, 59]. The key idea of this approach is that a *Green function* $G(r,r')$ can be defined as the solution of the equation

$$[1 - L_f^2\nabla_r^2]G(r,r') = \delta(r - r'),$$

where $\delta(r-r')$ is the Dirac δ-function. The Green function can be interpreted as the probability that a neutron produced at r' will be absorbed at r. As shown in the references cited above, this function G can be used to replace the differential equation (2.9) and its boundary conditions (2.11) and (2.12) by an equivalent equation

$$\phi(r) = \frac{1}{k^{\text{eff}}}\int_X G(r,r')k^\infty(r')\phi(r')\,dr'. \tag{2.14}$$

There are several ways to compute G (see *e.g.*, the related remarks in [28, 118]). Different approaches are described in *e.g.*, [38] and [59]. The freedom in how to approximate $G(r,r')$ and how to derive from $G(r,r')$ a discrete version of G enables a much rougher discretization than with direct discretization of (2.9), provided that some effort is spent in finding good values for the discretized version of G. In this thesis, the method described in [38] is used.

For convenience, the discretization is chosen such that the discretization points coincide with the nodes in the core. Suppose that the core consists of I nodes. This leads to the probabilities $G_{i,j}$, $i,j = 1,\cdots,I$:

- $G_{i,j}$: the probability that a neutron produced in node j will be absorbed in node i.

The infinite multiplication factor k^{∞} and flux ϕ also are averaged in a nodal way. The discrete versions of k^{eff}, k^{∞} and ϕ then are defined as[4]:

- k_t^{eff}: The effective multiplication factor at time t.
- $k_{i,t}^{\infty}$: The average infinite multiplication factor in node i at time t.
- $\phi_{i,t}$: The average neutron flux in node i at time t.

The integral equation (2.14) results in the set of equations

$$\phi_{i,t} = \frac{1}{k_t^{\text{eff}}} \sum_{j=1}^{I} G_{i,j} k_{j,t}^{\infty} \phi_{j,t}, \quad i = 1,\cdots,I,\ t = 1,\cdots,T. \tag{2.15}$$

The discretized form of the fixed power level constraint (2.13) now reads

$$\sum_{i=1}^{I} k_{i,t}^{\infty} \phi_{i,t} = \frac{\nu P_{\text{c}}}{\mu_{\text{f}}(\Sigma_{\text{a}}^{(1)} + \Sigma_{\text{s}}^{(1,2)})}, \quad t = 1,\cdots,T. \tag{2.16}$$

2.1.5 Fuel burnup

In Section 2.1.3, time-independent cross sections were introduced. This enabled us to use separation of variables to find a steady-state estimation for the flux, as being the solution of a time-independent eigenvalue differential equation. However, fission and absorption processes cause a change in the material composition which will influence the neutron flux on the long term. This time-dependence will be described in the current section.

During reactor operation, the fission reactions cause burnup of the fuel. The amount of fissionable material, and therefore the fission cross-sections $\Sigma_{\text{f}}^{(1)}$ and $\Sigma_{\text{f}}^{(2)}$ decrease. In our current $1\,^{1}\!/_{2}$ group approximation it is assumed that the fast neutron removal cross section remains approximately constant. This is a reasonable assumption, since most of the neutron removal is caused by slowing down, and hence downscattering to the thermal group, which is determined

---

[4] Note that we introduce a time step subscript t here. Although this is not yet used right now, we will need it within two pages.

by the moderator instead of the fuel. Moreover, it is assumed that the thermal absorption cross section remains approximately constant. This cross section decreases, because of the decreasing atom density of fissionable nuclides, but it also increases, since the fission products absorb neutrons, without causing a fission or scattering reaction.

The decrease of the fission cross sections is determined by the decrease of the density N of fissionable nuclides:

$$\begin{aligned}\frac{dN(t)}{dt} &= -\sigma^{\text{fuel}}_{\text{a},(1)}N(t)\phi^{(1)}(t) - \sigma^{\text{fuel}}_{\text{a},(2)}N(t)\phi^{(2)}(t) \\ &= -\left(\sigma^{\text{fuel}}_{\text{a},(1)} + \sigma^{\text{fuel}}_{\text{a},(2)}\frac{\Sigma^{(1,2)}_{\text{s}}}{\Sigma^{(2)}_{\text{a}}}\right)N(t)\phi(t) \\ &= -\sigma^{\text{fuel}}_{\text{a}}N(t)\phi(t), \qquad (2.17)\end{aligned}$$

where $\sigma^{\text{fuel}}_{\text{a}}$ was defined on the fly. Since $\Sigma=\sigma N$, where σ is time-independent it follows that (the time dependence is left out for better readability):

$$\begin{aligned}\frac{d\Sigma^{(1)}_{\text{f}}}{dt} &= -\sigma^{\text{fuel}}_{\text{a}}\Sigma^{(1)}_{\text{f}}\phi, \\ \frac{d\Sigma^{(2)}_{\text{f}}}{dt} &= -\sigma^{\text{fuel}}_{\text{a}}\Sigma^{(2)}_{\text{f}}\phi. \qquad (2.18)\end{aligned}$$

From the definition (2.8) of k^∞, the change in time of k^∞ now can be found:

$$\begin{aligned}\frac{dk^\infty}{dt} = \frac{\nu^{(1)}\frac{d\Sigma^{(1)}_{\text{f}}}{dt} + \nu^{(2)}\frac{d\Sigma^{(2)}_{\text{f}}}{dt}\cdot\frac{\Sigma^{(1,2)}_{\text{s}}}{\Sigma^{(2)}_{\text{a}}}}{\Sigma^{(1)}_{\text{a}} + \Sigma^{(1,2)}_{\text{s}}} &= -\sigma^{\text{fuel}}_{\text{a}}\frac{\nu^{(1)}\Sigma^{(1)}_{\text{f}} + \nu^{(2)}\Sigma^{(2)}_{\text{f}}\frac{\Sigma^{(1,2)}_{\text{s}}}{\Sigma^{(2)}_{\text{a}}}}{\Sigma^{(1)}_{\text{a}} + \Sigma^{(1,2)}_{\text{s}}}\phi \\ &= -\sigma^{\text{fuel}}_{\text{a}}k^\infty\phi. \qquad (2.19)\end{aligned}$$

This resulting differential equation has to be supplied with a boundary condition in time. If the complete configuration of the fuel at the beginning of a cycle (BoC) is known, the initial value of k^∞ can be denoted by k^{BoC}_i:

$$k^\infty_{i,0} = k^{BoC}_i. \qquad (2.20)$$

In this simplest case no boundary conditions at the End of the Cycle (EoC) are needed. If equilibrium cycles are studied, the boundary conditions become more complex. In that case, a part of the core is filled with fresh fuel, but $k^\infty_{i,0}$ in the remaining bundles depends on $k^\infty_{j,t_{EoC}}$, where j denotes the position of the specific fuel bundle in the previous cycle:

$$k^\infty_{i,0} = \begin{cases} k^{\text{fresh}} & \text{when the fuel at position } i \text{ is fresh;} \\ k^\infty_{j,t_{EoC}} & \text{when fuel is moved from } j \text{ to } i \text{ at EoC.}\end{cases} \qquad (2.21)$$

For numerical use, the differential equation (2.19) has to be discretized. Possible choices here are forward discretization:

$$k^\infty_{i,t+1} - k^\infty_{i,t} = -\sigma^{\text{fuel}}_{\text{a}}k^\infty_{i,t}\phi_{i,t}\Delta_t \qquad (2.22)$$

and central discretization:

$$k^{\infty}_{i,t+1} - k^{\infty}_{i,t} = -\sigma_{\mathrm{a}}^{\mathrm{fuel}} \frac{k^{\infty}_{i,t}\phi_{i,t} + k^{\infty}_{i,t+1}\phi_{i,t+1}}{2} \Delta_t. \tag{2.23}$$

Central discretization leads to a larger number of nonlinear terms. On the other hand, it is much more accurate and stable. Both types of discretizations will be discussed in more detail in Section 5.2.6.

In order to describe the boundary conditions (2.20) and (2.21), a description of the reloading operation is necessary. This is postponed to Section 2.2, that deals with the aspects of the model that are more directly related to the optimization part. For the rest, our model is now complete. In the remaining subsections of the current section, we briefly describe some extensions to this basic model that are used in our work, and we give an overview of possible further extensions.

2.1.6 Burnable absorbers

The model as described above is suitable to test the applicability of different types of optimization algorithms. It does not take into account a number of extensions that are needed in real-life reactor calculations, however. One of the most important extensions is the addition of burnable poisons (BP), which will be introduced in this section.

When starting a new reactor cycle, the difference in reactivity between the freshly added bundles and the older bundles is quite large. Later on during the cycle, this difference becomes less and less pronounced, since the fresh bundles also have the highest burnup rate. The difference at BoC causes a large power concentration in the fresh bundles and their direct neighbors. Since there is a safety limit on the maximum local power, this means that there may be loading patterns which have a good objective function, but still cannot be applied, because the power peak constraints at BoC are violated.

This problem is solved by adding neutron absorbing material (burnable poison, BP) to fresh fuel bundles, that initially absorbs many neutrons, but burns up very fast, so that it does not affect the later part of the cycle. The high neutron absorption cross section of this material causes a higher total neutron removal rate, and hence it lowers k^{∞}. Ideally, this will reduce the flux in the bundle and in its vicinity, hence it reduces the power peak. The effect of adding BP is illustrated in Figure 2.2, which shows the difference of the power shape in each bundle of an example core without and with BP.

The addition of BP can be modeled by introducing some extra variables in the model. Let

$$\Sigma^{\mathrm{p}}_{\mathrm{a},i,t}, \; i = 1, \cdots, I, \;\; t = 1, \cdots, T$$

be the neutron absorption cross section of the burnable absorbers, which only plays a role for the thermal group. The infinite multiplication factor k^{∞} now depends not only on 'standard' fission and absorption cross sections, but also on these absorption cross sections of the burnable absorbers.

In order to separate the effect of the BP and the other elements in the fuel, we introduce $\overline{k}^{\infty}$ as a variable describing the (theoretical) infinite multiplication factor of the fuel in the case the

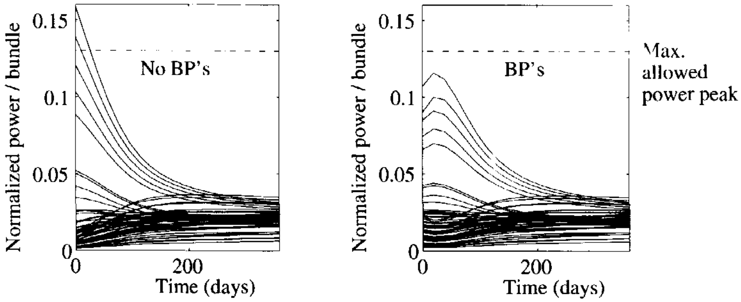


FIGURE 2.2: Power as function of time for each bundle, without and with burnable poisons added.

BP were ignored. Its description in terms of cross sections is therefore the same as the previous description of k^∞, but it is only an artificial variable that helps in determining the real infinite multiplication factor. The infinite multiplication factor itself is defined by using the definition of k^∞ in terms of cross sections as given by (2.8), where the neutron absorption cross section of BP is included:

$$k^\infty_{i,t} = \frac{\nu^{(1)}\Sigma^{(1)}_{\mathrm{f},i,t} + \nu^{(2)}\Sigma^{(2)}_{\mathrm{f},i,t}\dfrac{\Sigma_{\mathrm{s}}^{(1,2)}}{\Sigma_{\mathrm{a}}^{(2)} + \Sigma^{\mathrm{P}}_{\mathrm{a},i,t}}}{\Sigma_{\mathrm{a}}^{(1)} + \Sigma_{\mathrm{s}}^{(1,2)}}.$$

Now we use the fact that the fission cross section of fast neutrons Σ_{f} is negligible compared to the fission cross section of thermal neutrons. This is used to relate the 'old' $\overline{k}^\infty$' and the 'new' k^∞:

$$k^\infty_{i,t} \approx \frac{\nu^{(1)}\Sigma^{(1)}_{\mathrm{f},i,t} + \nu^{(2)}\Sigma^{(2)}_{\mathrm{f},i,t}\dfrac{\Sigma_{\mathrm{s}}^{(1,2)}}{\Sigma_{\mathrm{a}}^{(2)}}}{\Sigma_{\mathrm{a}}^{(1)} + \Sigma_{\mathrm{s}}^{(1,2)}} \cdot \frac{1}{1+\Sigma^{\mathrm{P}}_{\mathrm{a},i,t}/\Sigma_{\mathrm{a}}^{(2)}} = \frac{\overline{k}^\infty_{i,t}}{1+\Sigma^{\mathrm{P}}_{\mathrm{a},i,t}/\Sigma_{\mathrm{a}}^{(2)}}. \tag{2.24}$$

Thus, the effect of the BP in the infinite multiplication factor is reduced to one product term, and the infinite multiplication factor k^∞ can be calculated at any time by using the values of $\overline{k}^\infty$ and $\Sigma^{\mathrm{P}}_{\mathrm{a}}$.

The BP absorption cross section decreases in time with a factor depending on neutron flux, just as the fission cross sections (2.18):

$$\frac{d\Sigma^{\mathrm{P}}_{\mathrm{a}}}{dt} = -\sigma^{\mathrm{P}}_{\mathrm{a}}\Sigma^{\mathrm{P}}_{\mathrm{a}}\phi. \tag{2.25}$$

Combining the elements we have derived above, we arrive at a model where k^∞ is defined as

$$k^\infty_{i,t} = \overline{k}^\infty_{i,t}\frac{1}{1+\Sigma^{\mathrm{P}}_{\mathrm{a},i,t}/\Sigma_{\mathrm{a}}^{(2)}} \tag{2.26}$$

while $\overline{k}^\infty_{i,t}$ and $\Sigma^{\mathrm{P}}_{\mathrm{a},i,t}$ are defined using either the forward discretizations (*cf.* (2.22)):

$$\overline{k}^\infty_{i,t+1} - \overline{k}^\infty_{i,t} = -\sigma_{\mathrm{a}}^{\mathrm{fuel}}\overline{k}^\infty_{i,t}\phi_{i,t}\Delta_t$$

$$\Sigma^{\mathrm{p}}_{\mathrm{a},i,t+1} - \Sigma^{\mathrm{p}}_{\mathrm{a},i,t} = -\sigma^{\mathrm{p}}_{\mathrm{a}} \Sigma^{\mathrm{p}}_{\mathrm{a},i,t} \phi_{i,t} \Delta_t \tag{2.27}$$

or the central discretizations, see (2.23). The boundary conditions for $\overline{k}^{\infty}$ are the same as in (2.20) and (2.21). The boundary conditions for $\Sigma^{\mathrm{p}}_{\mathrm{a},i,t}$ are slightly more complicated, since the initial BP concentration is a decision parameter. Given the cross section $\Sigma^{\mathrm{p}}_{\mathrm{a,max}}$, corresponding to the maximum allowed BP concentration, they can be stated as

$$\Sigma^{\mathrm{p}}_{\mathrm{a},i,0} = \begin{cases} \in [0, \Sigma^{\mathrm{p}}_{\mathrm{a,max}}] & \text{when the fuel at position } i \text{ is fresh;} \\ \Sigma^{\mathrm{p}}_{\mathrm{a},j,t_{EoC}} & \text{when fuel is moved from } j \text{ to } i \text{ at EoC.} \end{cases}$$

In practice, the second case can be ignored since the BP concentration at EoC is negligible compared to the overall accuracy of the model. The exact mathematical formulation of the equation will be postponed until we have a description of the variables describing the reload pattern.

2.1.7 Different types of nuclides

Until now, it was assumed that the different cross sections are determined by only one type of fissionable material, and that the absorption cross sections can be taken approximately constant. In order to have a more accurate description, we need to distinguish between several types of nuclides that interact with neutrons and with each other. This has several consequences. The absorption cross sections can no longer be assumed to be constant. Furthermore, the decay of k^{∞} can no longer be described in such a simple way as in (2.19), since almost all elements in the definition of k^{∞} (2.8) are time-dependent with different decay rates, including the terms in the denominator. Instead, a number of nuclide densities is defined

$$N^{q}_{i,t}, \quad q = 1, \cdots, Q,$$

where Q is the number of nuclides considered. A possible nuclide chain is given in Figure 2.3. As one can see from this chain, there are several positions in the chain where neutrons are captured, either to be absorbed or to induce a fission. The densities of the different nuclides are highly correlated. One sees, for example, that the ^{239}Pu (Plutonium) density increases due to

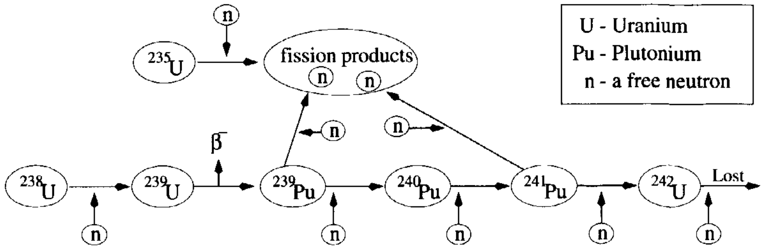


FIGURE 2.3: Nuclide Chain.

neutron capture in ^{238}U (Uranium), where it is assumed that ^{239}U immediately is transmuted into ^{239}Pu. The ^{239}Pu density decreases due to absorption of another neutron. In the same way as (2.17), the change of the ^{239}Pu density is given as

$$\frac{dN_i^{[9]}}{dt} := \sigma_{\mathrm{a}}^{[8]} N_i^{[8]} \phi_i - \sigma_{\mathrm{a}}^{[9]} N_i^{[9]} \phi_i, \tag{2.28}$$

where the superscript in brackets represents the last digit of the nuclide number, and $\sigma_{\mathrm{a}}^{[9]}$ covers both the fission cross section and the so called *breeding* to ^{240}Pu. In addition to the fissionable nuclides, one also has to take into account some fission products that have a large absorption cross section. The two most essential poisonous nuclides are ^{135}Xe (Xenon) and ^{149}Sm (Samarium). Since the concentrations of these nuclides reach equilibrium values in a relatively short time, their density can be computed explicitly, once the fission cross section and current neutron flux are known.

The final computation of k^∞ can no longer be done by implementing the burnup equation (2.19) directly. Instead we implement burnup equations like (2.28) for the different nuclides. Then, from the different nuclide densities, the needed macroscopic cross sections are computed in equations of the following form (only one specific absorption cross section for the fast group is given as an example):

$$\Sigma_{\mathrm{a},i,t}^{(1)} := \sum_{q=1}^{Q} \sigma_{\mathrm{a}}^{[q],(1)} N_{i,t}^{[q]}, \tag{2.29}$$

and finally k^∞ is related to these cross sections by its definition (2.8):

$$k_{i,t}^{\infty} = \frac{\nu^{(1)} \Sigma_{\mathrm{f},i,t}^{(1)} + \nu^{(2)} \Sigma_{\mathrm{f},i,t}^{(2)} \dfrac{\Sigma_{\mathrm{s},i,t}^{(1,2)}}{\Sigma_{\mathrm{a},i,t}^{(2)}}}{\Sigma_{\mathrm{a},i,t}^{(1)} + \Sigma_{\mathrm{s},i,t}^{(1,2)}}. \tag{2.30}$$

2.1.8 Other extensions

The current model, together with the extensions described in this chapter, gives a rather accurate description of the behavior of a reactor core. A truly accurate analysis should treat even more core specific elements. Some of these model extensions are described in this section.

In a real-life reactor core, some positions in the core are reserved for control rods. These are rods with a zero fission cross section, and a high absorption cross section, that can be moved vertically in the core to control the reactor power.

In Section 2.1.6, it was assumed that burnable absorbers are spread uniformly over the fuel. This is not necessary the case. Sometimes, they are assembled in thicker BP rods, so that the BP in the inner part of these rods only starts to play a role when the outer part gets 'saturated' with neutrons. This way, the absorption cross section of the whole rod is kept large for a longer time. Modeling this feature requires either more sophisticated modeling techniques or a finer discretization mesh with several mesh points within the BP stave.

Another extension is to model the 3D behavior of the core. As a matter of fact, the boundary effects at the top and the bottom have their influences on the whole core. Also partial insertion of control rods has major effects in the vertical dimension. A right choice of the cross section parameters in the 2D model may give a rather good prediction of the actual 3D behavior, but more accurate computations should deal with this extension.

A further extension is to use a finer nodal approach. Instead of one mesh point per fuel assembly, one might choose 4 (2 × 2) nodes. In doing so, a bundle can be placed in a node with 4 possible different rotations, thus giving an enormous increase of the number of loading patterns. For a really accurate calculation, one has to switch to a direct discretization of the diffusion equation, where 10 × 10 mesh points in 2D calculations are not uncommon, with possibly even more mesh points in vertical direction.

As can be seen from this list, the models as used in this paper are certainly not the most detailed ones. Nevertheless, they are suitable for our purposes. Our aim is not to evaluate a given core very accurately, but to find good loading patterns. A method is suitable for this purpose if it preserves the quality order of the different loading patterns. As is indicated by Figure 2.4, a nodal model is suited for this purpose. This picture shows that the power peak is underestimated by the nodal model that was used, so that the method may return solutions that are feasible in the nodal model, but not in accurate models. This is circumvented by evaluating a few rather different patterns with the detailed model before the optimization starts. These are used to estimate the 'drift' of the power peak in the nodal model, and the power peak constraints in the nodal model can be adjusted accordingly. After the nodal model has determined one or a few good loading patterns, a detailed model evaluation may be used to finally select the best one of these candidate patterns, possibly followed by a very limited local search performed using the more accurate model around these patterns.

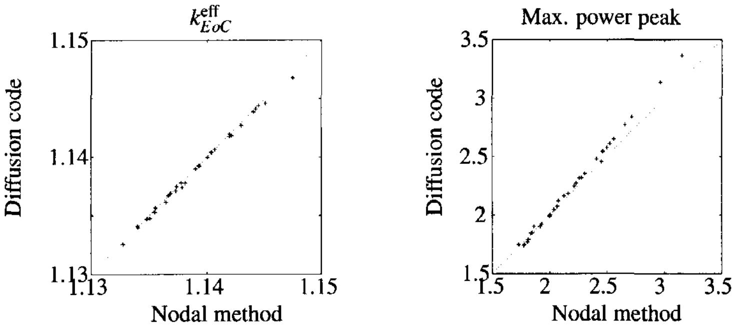


FIGURE 2.4: Comparison of k^{eff}_{EoC} (left) and the maximum power peak (right) on 34 test cores, obtained by a fast nodal method and a detailed diffusion method. These results are cited from [39].

2.2 Elements of the optimization models

In the previous section, we discussed the concepts from reactor-physics that are needed to build an optimization model. For the evaluation of a given core composition, this would be sufficient. In order to find an optimal loading pattern, more information is needed, and more variables have to be introduced. We define variables that specify the current loading pattern, and constraints are introduced such that these variables can only represent valid loading patterns. It must be defined how these loading pattern variables relate to the physical variables introduced in the previous section. We need an objective function that enables us to judge between patterns. It is also described how we can profit from symmetry in the reactor core.

Although an overview of existing solution methods is postponed until the next chapter, and our solution method will be treated even later, the model that will be discussed here is somewhat tailored to our solution method. It is unavoidable to make some choices at this stage, which are dependent on the solution method being used. Whenever this is the case, it will be made as clear as possible.

2.2.1 Loading pattern specification

In the cores that we will consider, an equilibrium cycle loading strategy is applied. At each EoC, the same number of fuel bundles is discharged, and all those removed bundles are of the same age, *i.e.*, they all have entered the core as fresh bundles and they have spent the same number of subsequent cycles in the core. Furthermore, the applied loading pattern will be exactly the same for many cycles, so that in the long term, ideally, each cycle will be exactly the same as the previous cycle.

To achieve such a scheme, each bundle has to be moved at each reloading, since if a node contained a 2 times reloaded bundle in some cycle, it also has to contain a 2 times reloaded bundle in the next cycle, so that the first bundle has to move to a position that is used to contain a 3 times reloaded bundle. A handy notation to describe such loading patterns is the trajectory notation [42]. For example, suppose we have 12 nodes in the core, and at each EoC, three bundles are discharged. We then may have the following three trajectories in the core:

$$\begin{array}{ccccccc} 4 & \rightarrow & 6 & \rightarrow & 8 & \rightarrow & 10, \\ 2 & \rightarrow & 3 & \rightarrow & 1 & \rightarrow & 5, \\ 7 & \rightarrow & 11 & \rightarrow & 9 & \rightarrow & 12, \end{array}$$

indicating for example that during a reloading, the bundle in node 4 is moved to node 6, and a fresh bundle is supplied in node 4. In order to be able to formulate the problem as an optimization problem, binary variables are used that describe the trajectories. We define

I: the number of nodes in the core;

L: the number of cycles that a bundle stays in the core before it will be discharged;

M: the number of trajectories in the core, *i.e.*, the number of fresh bundles that is entered into the core at each reloading. Note that $L \cdot M = I$.

Using these parameters, we introduce the set of variables

$$x_{i,\ell,m},\; i = 1,\cdots,I,\; \ell = 1,\cdots,L,\; m = 1,\cdots,M,$$

that are defined in the following way:

$$x_{i,\ell,m} = \begin{cases} 1 & \text{if node } i \text{ will contain the bundle of age } \ell \text{ from trajectory } m, \\ 0 & \text{otherwise.} \end{cases}$$

The index m is denoted as the 'bundle-number' of the bundle. The variables $x_{i,\ell,m}$ are simply assignment variables, restricted by the following assignment constraints:

$$\begin{aligned} \sum_{\ell=1}^{L} \sum_{m=1}^{M} x_{i,\ell,m} &= 1,\; i = 1,\cdots,I, \\ \sum_{i=1}^{I} x_{i,\ell,m} &= 1,\; \ell = 1,\cdots,L,\; m = 1,\cdots,M, \\ x_{i,\ell,m} &\in \{0,1\}. \end{aligned}$$

These constraints are sufficient to ensure that exactly one bundle is placed in each node, and each bundle is placed in exactly one node.

2.2.2 Equilibrium cycle condition

Once the assignment variables are defined, the physical behavior of the core has to be coupled with the actual loading pattern. That is, the boundary conditions (2.21) are to be defined more precisely. These conditions are split into two parts.

The part that is used when a bundle is fresh, is rather simple. It can be formulated as

$$k_{i,1}^{\infty} = \sum_{m=1}^{M} x_{i,1,m} k^{\text{fresh}}.$$

We assume here that all fresh bundles are of the same –given– composition, which is defined by the infinite multiplication factor k^{fresh}. Though in reality there is often a choice between a small number of different fresh fuel types, this is left out for simplicity. It would however not alter the model too much and in the burnable poison model (see below) we actually address the situation in which the burnable poison concentration per fresh bundle can be varied.

The second part of the boundary conditions is slightly more difficult. It handles the case that a bundle is not fresh. A certain $k_{i,1}^{\infty}$ has to be matched with exactly that $k_{j,T}^{\infty}$ that describes the bundle in the same trajectory, in the previous cycle. This is accomplished by the second part of the constraint:

$$k_{i,1}^{\infty} = \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} x_{j,\ell-1,m} k_{j,T}^{\infty}.$$

The aggregated constraint now is simply the sum of the above two parts:

$$k^{\infty}_{i,1} = \sum_{m=1}^{M} x_{i,1,m} k^{\text{fresh}} + \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} x_{j,\ell-1,m} k^{\infty}_{j,T}. \tag{2.31}$$

In the case that burnable poisons are considered, the fresh bundles are no longer described by one single parameter, since the BP concentration may be varied independent of k^{fresh}. Therefore, a new variable is introduced for each $m = 1, \cdots, M$:

$\Sigma^{\text{p,fresh}}_{\text{a},m}$: the cross section of BP in the fresh bundle of trajectory m.

Depending on the type of BP used, the fresh fuel BP concentration can take a (small) number of predetermined values, or its value can vary (almost) continuously between 0 and a specified maximum concentration.

In order to specify the reloading equations in this case, recall from Section 2.1.6 that in the BP model a distinction is made between $\overline{k}^{\infty}$, the (artificial) infinite multiplication factor where BP is neglected, and k^{∞}, the total multiplication factor that accounts for the presence of BP. The properties $\overline{k}^{\infty}$ and $\Sigma^{\text{p}}_{\text{a}}$ describe the properties of a certain bundle, the actual multiplication factor is derived via (2.26). So, the reloading equations are to be specified for $\overline{k}^{\infty}$ and $\Sigma^{\text{p}}_{\text{a}}$ only. The reloading equation for $\overline{k}^{\infty}$ is given by

$$\overline{k}^{\infty}_{i,1} = \sum_{m=1}^{M} x_{i,1,m} k^{\text{fresh}} + \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} x_{j,\ell-1,m} \overline{k}^{\infty}_{j,T}.$$

The burnable poison concentration has the property that it depletes very rapidly; at the end of the fuel cycle the fraction of the BP that is still active is smaller than the precision of the used data. Therefore it is safe to ignore the BP concentration in older bundles, so that the BP reloading equation is described by

$$\Sigma^{\text{p}}_{\text{a},i,1} = \sum_{m=1}^{M} x_{i,1,m} \Sigma^{\text{p,fresh}}_{\text{a},m}.$$

In the multi-nuclide model, the situation is even more different. A bundle is now described by its nuclide densities, hence a reloading equation has to be specified for each nuclide type:

$$N^{q}_{i,1} = \sum_{m=1}^{M} x_{i,1,m} N^{q,\text{fresh}} + \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} x_{j,\ell-1,m} N^{q}_{j,T}.$$

The cross sections and k^{∞} at BoC then are computed by equations of the form (2.29) and (2.30), respectively.

2.2.3 Quadrant and Octant symmetry

The smallest core that we have used contains 96 bundles. A typical reactor may contain a few hundred bundles. This leads to huge models. However, typical reactor cores have a nice symmetry. If we restrict to loading patterns that obey some symmetry rules, the number of possible loading patterns reduces drastically. In our equilibrium cycle model, ideal symmetry is obtained when each symmetric part contains a complete set of trajectories. If quadrant symmetry is used, then this is possible if the number of bundles in each quadrant is a multiple of the number of age groups in the core, and if there is no single center node in the core. Actually, our (imaginary) test cores are designed in such a way that this holds, *cf.* Figure 2.5. When using octant symmetry, it is already obvious from this figure that the same situation cannot hold, since there are bundles on the diagonal which are shared by two octants. In our model we added the artificial constraint that adjacent pairs of diagonal nodes share the same bundle, which is split into halves, by imposing the constraint[5]

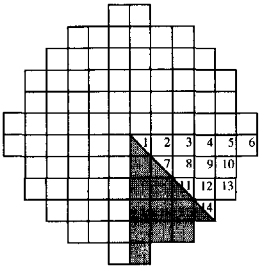


FIGURE 2.5: Octant symmetric core.

$$x_{i,\ell,m} = x_{j,\ell,m}, \quad \text{for all adjacent diagonal pairs } (i,j),\ \ell = 1,\cdots,L,\ m = 1,\cdots,M.$$

This introduces a small bias, since we assume that the 're-assembled' bundle at EoC has the average properties of the two diagonal nodes.

In the model formulation, this division of the diagonal bundles makes it necessary to take into account the volumes V_i of the nodes. For example, the power normalization constraint (2.16) now is formulated as

$$\sum_{i=1}^{I} V_i k_{i,t}^{\infty} \phi_{i,t} = \frac{\nu P_c}{\mu_f(\Sigma_a^{(1)} + \Sigma_s^{(1,2)})}, \quad t = 1,\cdots,T \tag{2.32}$$

where

$$V_i = \begin{cases} {}^1\!/_2 & \text{if node } i \text{ is a diagonal node,} \\ 1 & \text{otherwise.} \end{cases}$$

The other constraints are adapted in a similar way.

---

[5] In our actual implementations, the $x_{j,\ell,m}$-variables are eliminated in before and all occurrences of $x_{j,\ell,m}$ are replaced by $x_{i,\ell,m}$. Since this messes up the notation, this replacement is not used in the models in the thesis.

2.2.4 Safety restriction

The power in a reactor core must be spread quite evenly over the core. Although some variations are unavoidable, the local power density in a bundle may not become too high for safety reasons. We introduce the *power peaking factor*, f^{lim}, which is the maximum allowed ratio between the maximum and the average local power peak. The average power peak can be derived from (2.32) by dividing the right hand side by the sum of the volumes. The power peaking constraints then are defined as

$$k^{\infty}_{i,t}\phi_{i,t} \le \frac{f^{\text{lim}}}{\sum\limits_{i=1}^{I} V_i} \cdot \frac{\nu P_{\text{c}}}{\mu_{\text{f}}(\Sigma_{\text{a}}^{(1)} + \Sigma_{\text{s}}^{(1,2)})}, \quad i = 1,\cdots,I,\ t = 1,\cdots,T. \tag{2.33}$$

2.2.5 Objective function

In reloading pattern optimization literature different objective functions have been applied. It is possible to search for a loading pattern for which the cycle length is maximal before the reactor becomes subcritical. Another approach is to minimize the leakage out of the core. Here we have chosen to fix the cycle length, and search for a pattern for which the reactivity of the core at EoC is maximal. According to [132], all these possible objective functions are more or less equivalent in the case of an equilibrium cycle.

As a measure for the reactivity we use the uncontrolled effective multiplication factor k^{eff}:

$$\max k^{\text{eff}}_{t_{EoC}}.$$

The word *uncontrolled* refers to the case that the core is not controlled by external regulators, such as control rods. In actual core control, the effective multiplication factor is always kept around one.

In the implementation of an optimization algorithm, the objective may be adapted to contain some penalty. For example, in our implementation, it turned out to be advantageous to relax the safety limitation (2.33) with a small value ε, which is then penalized in the objective. This will be discussed later on.

2.3 Summary of the basic models

This chapter is concluded by an overview of the standard optimization models that are obtained by merging the results from this chapter. First, the basic $1^1\!/_2$ group model is stated. Next, this model is extended as to include burnable poison optimization. This section is concluded with an adaptation of the basic model. We came up with this modified model only in the final stage of the project. It alleviates some difficulties in the other models, arising from our solution methodology. Due to time limitations, we could not work out that model further, but some preliminary computational results for this model are presented in Chapter 6.

In the model listings, the flux variable ϕ is replaced by the normalized removal rate R. The unnormalized removal rate $\hat{R}$ is obtained by multiplying the flux with the removal probability:

$$\hat{R} = \phi \cdot (\Sigma_\mathrm{a}^{(1)} + \Sigma_\mathrm{s}^{(1,2)}).$$

This removal rate is scaled in order to simplify the model presentation:

$$R = \frac{\mu_\mathrm{f}}{\nu P_\mathrm{c}} \hat{R}.$$

This scaling, together with the substitution

$$\alpha = \sigma_\mathrm{a}^\mathrm{fuel} \frac{\nu P_\mathrm{c}}{\mu_\mathrm{f}(\Sigma_\mathrm{a}^{(1)} + \Sigma_\mathrm{s}^{(1,2)})} \tag{2.34}$$

makes the presentation of the problem much easier.

2.3.1 *The basic $1^1/_2$ group model*

$$\max_{x,k^\infty,R,k^\mathrm{eff}} \; k_T^\mathrm{eff}$$

subject to:

the linear constraints:

$$\sum_{i=1}^{I} V_i x_{i,\ell,m} = 1, \qquad \ell = 1,\cdots,L,\ m = 1,\cdots,M \tag{2.35}$$

$$\sum_{\ell=1}^{L} \sum_{m=1}^{M} x_{i,\ell,m} = 1, \qquad i = 1,\cdots,I \tag{2.36}$$

the nonlinear constraints:

$$\begin{aligned} k_{i,1}^\infty &= \sum_{m=1}^{M} x_{i,1,m} k^\mathrm{fresh} \\ &\quad + \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} V_j x_{j,\ell-1,m} k_{j,T}^\infty, \qquad i = 1,\cdots,I \end{aligned} \tag{2.37}$$

$$k_t^\mathrm{eff} R_{i,t} = \sum_{j=1}^{I} G_{i,j} k_{j,t}^\infty R_{j,t}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T \tag{2.38}$$

$$\sum_{i=1}^{I} V_i k_{i,t}^\infty R_{i,t} = 1, \qquad t = 1,\cdots,T \tag{2.39}$$

$$k_{i,t+1}^\infty = k_{i,t}^\infty - \alpha \Delta_t k_{i,t}^\infty R_{i,t}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T-1 \tag{2.40}$$

$$k^{\infty}_{i,t}R_{i,t} \leq \frac{f^{\lim}}{\sum_{j=1}^{I} V_j}, \qquad i=1,\cdots,I,\ t=1,\cdots,T \tag{2.41}$$

the variable bounds:

$$k^{\text{eff}}_{t} \geq 0, \qquad t=1,\cdots,T \tag{2.42}$$

$$k^{\infty}_{i,t}, R_{i,t} \geq 0, \qquad i=1,\cdots,I,\ t=1,\cdots,T \tag{2.43}$$

$$x_{i,\ell,m} = x_{j,\ell,m}, \qquad \text{all adjacent diagonal pairs } (i,j),\ \ell=1,\cdots,L,\ m=1,\cdots,M \tag{2.44}$$

$$x_{i,\ell,m} \in \{0,1\}, \qquad i=1,\cdots,I,\ \ell=1,\cdots,L,\ m=1,\cdots,M. \tag{2.45}$$

There are several ways to look at this model. Given an assignment of the x variables that satisfies the assignment constraints (2.35)–(2.36), there exists exactly one nonnegative assignment of the variables k^{∞}, R and k^{eff} that satisfies the nonlinear system (2.37)–(2.40). This assignment can be calculated by iterative methods and may or may not satisfy the power peaking constraint. Local search methods and other black box optimization approaches make use of this feature: the equality constraints are left out of the actual optimization procedure, and they are only used to calculate the objective function and the value of the power peaking constraints for fixed $x_{i,\ell,m}$-variables by iterative methods. Since the resulting objective function and power peaking constraints are nonconvex functions of the $x_{i,\ell,m}$-variables, it is very difficult, and for realistic size problems practically impossible, to guarantee that a found optimum is globally optimal.

Another approach is to treat this model as a mixed-integer nonlinear optimization (MINLP) model. In this approach, all equalities are used as constraints in the optimization model. In that way, we are able to use derivative information from all equalities and constraints. The model is nonconvex, as the equality constraints and the power peaking constraint contain terms that are bi- and trilinear in the variables. This nonconvexity makes it very difficult to find global optimal solutions.

For both approaches, the local optimal solution that will be found depends on the starting points from which the search is initiated. Since the power peaking constraint is nonconvex as well, algorithms may erroneously conclude that there is no feasible loading pattern at all. Therefore, care has to be taken and several adaptations to the model have to be made in order to guarantee stability of algorithms. The aim of our work is to find stable implementations following the MINLP approach.

The model is an extension to the model of de Klerk *et al.* [65] mainly in three ways. First of all, the model in [65] had k^{eff} fixed to 1, and instead the burnup of the bundles was introduced as an optimization variable. Secondly, there was no distinction between the bundles of the same age. However, it is clear that a fresh bundle in the center has much different characteristics at EoC than a fresh bundle at the boundary, so this was really a rough estimation. Thirdly, there was no division of the cycle into time-steps, but the power distribution was assumed to be constant during the whole cycle.

2.3.2 Model including Burnable Poisons

In this case, a bundle is described by the two properties $\overline{k}^{\infty}$ and $\Sigma_{\mathrm{a}}^{\mathrm{p}}$, and in each bundle and in each time step k^{∞} is derived from these quantities. Besides α as defined in (2.34), we also define

$$\alpha^{\mathrm{p}} = \sigma_{\mathrm{a}}^{\mathrm{p}} \frac{\nu P_{\mathrm{c}}}{\mu_{\mathrm{f}}(\Sigma_{\mathrm{a}}^{(1)} + \Sigma_{\mathrm{s}}^{(1,2)})}. \tag{2.46}$$

This leads to the following model:

$$\max_{x,\overline{k}^{\infty},\Sigma_{\mathrm{a}}^{\mathrm{p}},\Sigma_{\mathrm{a}}^{\mathrm{p,fresh}},k^{\infty},R,k^{\mathrm{eff}}} \quad k_T^{\mathrm{eff}}$$

subject to:

the linear constraints:

$$\sum_{i=1}^{I} V_i x_{i,\ell,m} = 1, \qquad \ell = 1,\cdots,L,\ m = 1,\cdots,M \tag{2.47}$$

$$\sum_{\ell=1}^{L}\sum_{m=1}^{M} x_{i,\ell,m} = 1, \qquad i = 1,\cdots,N \tag{2.48}$$

the nonlinear constraints:

$$\begin{aligned}\overline{k}_{i,1}^{\infty} = {} & \sum_{m=1}^{M} x_{i,1,m} k^{\mathrm{fresh}} \\ & + \sum_{\ell=2}^{L}\sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} V_j x_{j,\ell-1,m} \overline{k}_{j,T}^{\infty}, \qquad i = 1,\cdots,N\end{aligned} \tag{2.49}$$

$$\Sigma_{\mathrm{a},i,1}^{\mathrm{p}} = \sum_{m=1}^{M} x_{i,1,m} \Sigma_{\mathrm{a},m}^{\mathrm{p,fresh}}, \qquad i = 1,\cdots,N \tag{2.50}$$

$$(1 + \Sigma_{\mathrm{a},i,t}^{\mathrm{p}}/\Sigma_{\mathrm{a}}^{(2)}) k_{i,t}^{\infty} = \overline{k}_{i,t}^{\infty}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T \tag{2.51}$$

$$k_t^{\mathrm{eff}} R_{i,t} = \sum_{j=1}^{I} G_{i,j} k_{j,t}^{\infty} R_{j,t}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T \tag{2.52}$$

$$\sum_{i=1}^{I} V_i k_{i,t}^{\infty} R_{i,t} = 1, \qquad t = 1,\cdots,T \tag{2.53}$$

$$\overline{k}_{i,t+1}^{\infty} = \overline{k}_{i,t}^{\infty} - \alpha \Delta_t \overline{k}_{i,t}^{\infty} R_{i,t}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T-1 \tag{2.54}$$

$$\Sigma_{\mathrm{a},i,t+1}^{\mathrm{p}} = \Sigma_{\mathrm{a},i,t}^{\mathrm{p}} - \alpha^{\mathrm{p}} \Delta_t \Sigma_{\mathrm{a},i,t}^{\mathrm{p}} R_{i,t}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T-1 \tag{2.55}$$

$$k_{i,t}^{\infty} R_{i,t} \leq \frac{f^{\mathrm{lim}}}{\sum_{j=1}^{I} V_j}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T \tag{2.56}$$

the variable bounds:

$$
\begin{aligned}
k_t^{\text{eff}} &\geq 0, & t &= 1,\cdots,T & (2.57)\\
\overline{k}_{i,t}^{\infty}, \Sigma_{\text{a},i,t}^{\text{p}}, k_{i,t}^{\infty}, R_{i,t} &\geq 0, & i &= 1,\cdots,I,\ t = 1,\cdots,T & (2.58)\\
\Sigma_{\text{a},m}^{\text{p,fresh}} &\in [0, \Sigma_{\text{a,max}}^{\text{p}}], & m &= 1,\cdots,M & (2.59)\\
x_{i,\ell,m} &= x_{j,\ell,m}, & &\text{all adjacent diagonal pairs } (i,j), & (2.60)\\
& & &\ell = 1,\cdots,L,\ m = 1,\cdots,M & \\
x_{i,\ell,m} &\in \{0,1\}, & i &= 1,\cdots,I,\ \ell = 1,\cdots,L,\ m = 1,\cdots,M. & (2.61)
\end{aligned}
$$

Although this model has some extra variables and constraints, the structural difference when compared to the previous model is very moderate. One real difference is that in the black box approach, the decision variables are not only the $x_{i,\ell,m}$-variables, but also the $\Sigma_{\text{a},m}^{\text{p,fresh}}$-variables. These variables may be continuous instead of binary, which can cause difficulties in some local search algorithms.

In the MINLP model, the nonlinearities are still bi- and trilinear terms. The increase of this type of constraints may decrease the stability of some algorithms, causing some extra work to get the implementation robust. The continuity of the $\Sigma_{\text{a}}^{\text{p,fresh}}$ variables is not a problem here, since continuous optimization is already involved in finding the values of the physical variables.

2.3.3 A new model

Although the model of Section 2.3.1 is exact for integer solutions, it is not completely accurate for fractional solutions, *i.e.*, solutions where $x_{i,\ell,m}$ values are not exactly 0 or 1, but some value in between. Although this is not a problem for optimization methods that use only integer values of $x_{i,\ell,m}$, it may cause problems when continuous optimization is used to solve the problem. The inaccuracy that may occur is illustrated by Figure 2.6. In this figure, we consider two fuel trajectories A and B. Fresh fuel is assigned to node 1 and 2, and the one year old bundle of trajectory B is placed in node 3. Node 4 and node 5 are each partly filled with the one year

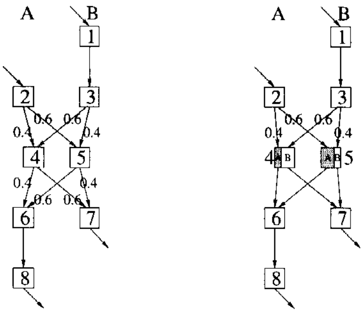


FIGURE 2.6: Fractional bundle reloading, wrong (left) and correct (right).

old bundle of trajectory A and for the other part with the two years old bundle of trajectory B (*i.e.*, $x_{4,2,A} = x_{5,3,B} = 0.4$ and $x_{4,3,B} = x_{5,2,A} = 0.6$). Now node 6 should contain the two years old bundle of trajectory A (*i.e.*, $x_{6,3,A} = 1$) and similarly, $x_{7,4,B} = 1$. But this is not the case. Using the reloading equation from the model in Section 2.3.1, it is illustrated at the right side of Figure 2.6 that node 6 contains fraction 0.4 of the *mixed* bundle in node 4 and fraction 0.6 of the *mixed* bundle in node 5. This results in a fraction 0.52 of the bundle that came from node 2 and a fraction 0.48 from the bundle that came from node 3. The statement $x_{6,3,A} = 1$ is thus interpreted incorrectly by the model.

In order to get a correct model, it is necessary to trace the fractions of different bundles in a node separately. This is accomplished by introducing the variable

$$\hat{k}^{\infty}_{i,\ell,m,t},$$

being the infinite multiplication factor at time t of the fraction of the bundle of age ℓ from trajectory m that resides in node i. In order to trace this fraction, the burnup has to be computed for each fraction separately at each time step. In the reloading equation, it is then possible to reload the correct fractions into the correct nodes. Returning to Figure 2.6, given that $x_{6,3,A} = 1$, it is possible to build $k^{\infty}_{6,1}$ as the weighted average of $\hat{k}^{\infty}_{4,2,A,T}$ and $\hat{k}^{\infty}_{5,2,A,T}$. Note that the right picture is still not completely correct, since during the cycle, the fractions in bundles 4 and 5 should be completely mixed. Only during reloading are they separated again into the parts denoted by 'A' and 'B'.

In the model described in Section 2.3.1, the reload equation (2.37) should be replaced by

$$\begin{aligned}
\hat{k}^{\infty}_{i,1,m,1} &= x_{i,1,m}k^{\text{fresh}}, && i = 1,\cdots,I,\ m = 1,\cdots,M \\
\hat{k}^{\infty}_{i,\ell,m,1} &= x_{i,\ell,m}\sum_{j=1}^{I} V_j\,\hat{k}^{\infty}_{j,\ell-1,m,T}, && i = 1,\cdots,I,\ \ell = 2,\cdots,L,\ m = 1,\cdots,M.
\end{aligned}$$

The burnup equation is then replaced by

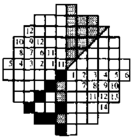
$$\begin{aligned}
\hat{k}^{\infty}_{i,\ell,m,t+1} &= \hat{k}^{\infty}_{i,\ell,m,t} - \alpha\Delta_t\,\hat{k}^{\infty}_{i,\ell,m,t+1}R_{i,t}, && i = 1,\cdots,I,\ \ell = 1,\cdots,L, \\
& && m = 1,\cdots,M,\ t = 1,\cdots,T-1.
\end{aligned}$$

Finally, a relation between all the separate $\hat{k}^{\infty}_{i,\ell,m,t}$ variables and the node-average $k^{\infty}_{i,t}$ is given:

$$k^{\infty}_{i,t} = \sum_{\ell=1}^{L}\sum_{m=1}^{M}\hat{k}^{\infty}_{i,\ell,m,t}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T.$$

In the BP model of Section 2.3.2 the modification is done in a similar way. There, the reload equation and the burnup equation for the variables $\overline{k}^{\infty}_{i,t}$ can be substituted by the corresponding equations for $\hat{k}^{\infty}_{i,\ell,m,t}$ as stated above. The variable $\overline{k}^{\infty}$ can be deleted completely when in (2.51) the variable $\overline{k}^{\infty}_{i,t}$ is substituted by $\sum_{\ell=1}^{L}\sum_{m=1}^{M}\hat{k}^{\infty}_{i,\ell,m,t}$.

Due to time limitations, we had not much opportunity to experiment with this modified model. Some encouraging results are discussed in Chapter 6 on page 138.



Chapter 3

Existing optimization approaches

Starting from the sixties, many authors have been working on the loading pattern optimization problem. A large variety of methods have been applied. It was recognized that it is very difficult to obtain globally optimal solutions, since the models exhibit a structure that makes the problem hard in a mathematical sense. Apart from the mathematical difficulties, there is also no unanimous understanding about what objective function should be selected, which makes it indeed very difficult to state that some loading pattern is globally optimal. To make it even worse, every author and every paper uses its own model, with its own scope, properties and simplifications. Thus comparison of results is only possible on a 'local scale', usually the scale of just one paper.

This chapter presents an overview of the literature, with emphasis on the used mathematical optimization methods. Remarks about the used models, their complexity or their scope will only be made on a rather generic level. The different optimization approaches are handled in more detail.

A main distinction between the used algorithms is the way in which they treat the optimization problem. The reload pattern optimization problem can be viewed as a mathematical optimization problem with two types of variables:

1. Decision variables specifying the fuel assignment (which bundle in which node, in which rotational position, and which BP concentration is included in each node);
2. Dependent variables specifying the core reactivity and depletion.

The variables are restricted by the following types of constraints:

1. Constraints on the decision variables, that specify which combinations of these variables are valid loading patterns;
2. Equations that describe how the dependent variables depend on the decision variables;
3. Operational constraints, either on the decision variables (engineering constraints), on the dependent variables (power peak constraints), or possibly on both types of variables together.

The depletion equations are in this description contained in the second type of constraints. Many algorithms consider only the decision variables in the actual optimization, while the dependent variables and the corresponding equations are hidden in a black box. This black box, which can include an arbitrary core depletion algorithm, returns the objective function value and safety limit values for fixed sets of decision variables, *i.e.,* for fixed loading patterns. In these methods, a (large) number of patterns is evaluated in a smart way, in order to find the best one. A number of these *direct search methods* is shortly described in Section 3.1.

Instead of treating the function evaluation for a fixed loading pattern as a black box (possibly opened a little bit when using perturbation theory), the black box can also be opened completely, and the information in it can be used in *derivative based* approaches to find search directions towards better loading patterns. Due to the nature of derivative based optimization methods, continuous relaxations are solved first, that may return fractional reloading patterns, and from there integer solutions are generated. Some variants based on this idea are explained in Section 3.2. Our method as described in the following chapter also falls into this category.

Perturbation theory (PT) is a tool that can be used in both types of methods. Local search methods use PT to predict the function value in some patterns without the need of completely evaluating these patterns. Some implementations of derivative based methods use PT to obtain gradient information or to linearize the problem. Another generic tool is the use of engineering constraints. PT and engineering constraints are discussed in Section 3.3.

The list of papers and methods below is not complete. The number of proposed methods is very large. Also, some methods are very similar to each other but have different names.

3.1 Direct search methods

In this section, a brief overview is given of those local search methods that are used to solve the loading pattern optimization problem.

Initially, one or more loading patterns must be given, which define the initial solution set. At each iteration, a neighborhood of the current (or parent) solution set is defined, containing a number of loading patterns that are related to the patterns in the current solution set in a way that may differ from method to method. Some patterns from this neighborhood are selected in either a deterministic or stochastic way, and their objective function is evaluated. After this evaluation, possibly some of these evaluated patterns are inserted into the current solution set, while other patterns might be removed from the current solution set. The decision, which patterns are to be inserted or removed from the solution set, is either deterministic or stochastic, and is usually based on the quality of the current solution set and the newly evaluated patterns The decision may also be influenced by the search history. The process is repeated until some stopping criterion is met, and the best pattern found is accepted as the (local) optimum pattern.

The class of local search methods is very wide, and moreover each method has many variants. Overviews of a number of different methods that are applicable to the reload pattern optimization problem are given by [15, 98]. These methods vary from commonly used methods (Simulated annealing (SA), Genetic algorithms (GA)), to methods with exotic names such as

Record-to-Record Travel, Great Deluge algorithm, and Population-Based Incremental Learning algorithm.

Since in local search methods the function evaluations are considered as a black box, these methods are applied in a number of other areas. Actually many combinatorial search methods are developed in rather different application areas, and there are very few common naming conventions. In current practice, the same names are used for methods that share some common basic property but otherwise are very different, while on the other hand totally different names are used for methods that are essentially the same, even within one application area [1, 51, 123].

Regarding this diversity of methods, the only goal of this section can be to present the general structure of the most common search methods that have been applied to reload pattern optimization. In Section 3.1.1 the Pairwise interchange algorithm will be discussed, which is the basic local search algorithm. Actually, as it will be described in Section 5.3.4, we designed an extension of this algorithm so that it can deal better with patterns that exceed the power peaking constraint. In the next three sections we discuss Simulated annealing, Evolutionary algorithms and Tabu search. Section 3.1.5 deals with some variants that are studied at the Delft University Interfaculty Reactor Institute within the scope of our joint research project [39].

3.1.1 Pairwise or Multiple Interchange

This is the basic variant of a local search algorithm. Although the method itself is very simple, it illustrates several aspects of local search heuristics that also appear in the other algorithms. The variants Pairwise Interchange (PI) and Multiple Interchange (MI) differ in the definition of the neighborhood only.

The (pairwise or multiple) interchange algorithm uses a single pattern as current solution. After evaluating all patterns in the neighborhood, the neighbor with the best objective function value is selected. If it is better than the objective function value of the current solution, the current solution is replaced by this neighbor solution. Otherwise, the algorithm terminates and returns the current solution as the (locally) optimum solution.

In PI, the neighborhood consists of all patterns that are obtained by exchanging two bundles in the current pattern. In k-Multiple Interchange, the neighbors are all patterns that are obtained by exchanging k bundles in the current pattern. The size of the neighborhood increases drastically for larger k. For $k = 2$ (which is PI), the number of neighbors is $I(I-1)/2$, where I is the number of bundles in the core. For $k > 2$, the number of neighbors is $O(I^k)$. For $k = I$ this is equivalent to a complete enumeration of all possible loading patterns! In practice, only the PI algorithm is applicable, unless the neighborhood is drastically reduced by additional restrictions.

The number of iterations in PI is not known in advance. Also, there is no guarantee that the algorithm will end with the best solution. It is very likely that the algorithm ends in a local optimum. The final solution also depends on the starting point, as is illustrated in Figure 3.1.

If the cost of evaluating the whole neighborhood in each iteration is too high, the neighborhood can be restricted at the risk of obtaining worse final solutions. One may for example restrict the neighborhood to exchanges of fuel bundles of the same (or adjacent) age, or to ex-

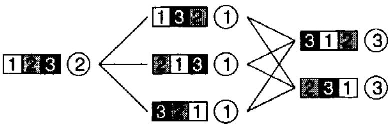


FIGURE 3.1: Pairwise Interchange tree of a core with three nodes. The bundles are indicated by gray values and numbers. The circled value is the objective value for the given pattern. Neighboring solutions in PI are connected. The two right solutions, with the best objective value, cannot be reached when starting from the left one.

changes of bundles in adjacent positions. Another smart way of reducing the neighborhood is the use of engineering constraints, which exclude fuel combinations that are known to be bad (Section 3.3.2). In the loading pattern optimization literature, we encountered such a restricted PI algorithm under the name Dynamic Programming [121]. Other PI implementations, sometimes combined with engineering constraints (see Section 3.3.2) are found in [29, 64, 83, 87].

First best neighbor selection

Computation time in PI may be restricted by not selecting the best of all neighbors, but simply the first neighbor encountered that improves the objective function. This typically will lead to less pattern evaluations, but to a worse final solution, It also introduces dependence on the order in which the neighbors are evaluated, as shown in the left illustration of Figure 3.2. Occasionally, this approach may lead to better quality solutions, as in the right illustration of Figure 3.2.

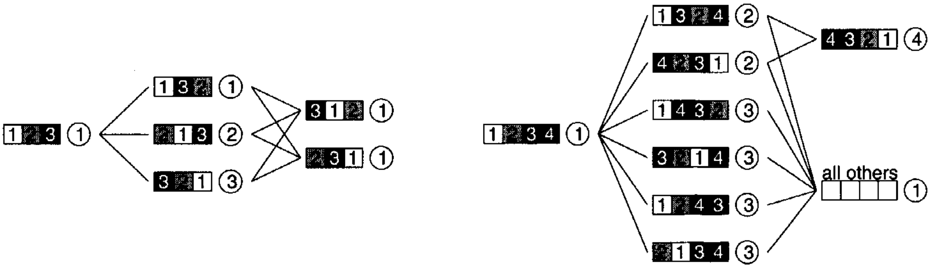


FIGURE 3.2: Two more Pairwise Interchange trees, starting with the solutions left in the trees as the parents. If the neighbors in the middle columns are evaluated in the order as listed, then the strategy of selecting the first improved neighbor as new parent leads to a worse final solution in the left tree, and to a better final solution in the right tree.

Handling infeasible patterns

There are different ways of handling infeasible patterns, that is, patterns violating some operational constraint such as a safety limitation. One might ignore all infeasible neighbors as soon as the first feasible solution is found. Another approach is to include the constraints in the objective function, multiplied with a penalty parameter. In this way, an infeasible pattern may be chosen only if it has a very good objective value with relatively small constraint violations, with the hope that this solution will have neighbors that are feasible and still have a good objective value. Since in this variant the current solution might be infeasible, while a feasible solution was found before, the best found feasible solution should be stored separately during the search. The quality of the results with this approach will depend on the value of the penalty parameter.

We have implemented a variant of this algorithm, which is described further in Section 5.3.4.

A weakness of the standard interchange algorithm is that it is unable to escape from a local optimum. This is because it never accepts a pattern with worse objective function than the current solution. Another weakness is the solution time required. In the standard PI algorithm, the number of neighbor evaluations per iteration grows quadratically in the number of nodes, while the evaluation time per neighbor increases, and also the average number of iterations before reaching a local optimal solution grows quite fast. The algorithms in the next sections are designed to reduce these problems.

3.1.2 Simulated Annealing

This algorithm allows the acceptance of parent solutions that are worse than the current solution, in order to make it possible to escape from local optima. These worse patterns are accepted with a certain probability, that depends on the optimization objective value, and on a 'temperature' T which slowly decreases during the search. The algorithm is based on the annealing process of a crystal. In this process, a liquid metal is cooled down in order to obtain a crystal. If the cooling is too fast, the resulting crystal is not at the minimum energy state, but it will be an amorphous structure at a higher energy state. Escaping from this energy state is not possible without addition of external energy, so it is a local optimum. Using slower cooling, the atoms have more opportunities to escape from local optima and reach lower energy states, and the resulting crystal will better resemble an ideal, regular crystal.

More precisely, the Simulated Annealing (SA) algorithm is stated as follows. Starting from a parent solution, a random child solution is chosen. If this child has better objective function than the parent, it is accepted as the new parent solution, otherwise it is accepted with a certain probability. This probability depends on the objective function and on the temperature parameter. After a number of such acceptances and rejections (a chain), the temperature parameter is decreased, which means a lower probability of selecting worse patterns. The algorithm ends when some stopping criterion is satisfied.

Let f_n be the objective value of the current solution, and f_{n+1} the objective value of the selected child. The acceptance probability for the child, for given temperature parameter T, is given as follows:

$$P(f_n, f_{n+1}, T) = \begin{cases} 1 & \text{if } f_{n+1} \text{ is better than } f_n \\ e^{\frac{-|f_n - f_{n+1}|}{T}} & \text{otherwise} \end{cases}$$

At the end of a chain, the temperature parameter is usually decreased by a fixed factor $\alpha < 1$:

$$T_{k+1} = \alpha T_k,$$

but the reduction formula may take more exotic forms [117]:

$$T_{k+1} = \frac{T_k}{1 + \frac{T_k \ln(1+\delta)}{3\sigma_k}}$$

where δ is some suitable parameter and σ_k is the standard deviation of the objective function values in the kth evaluation chain. The length of such a chain can be fixed in advance, or may depend on the quality of the obtained solutions.

The neighborhood from which trial patterns are chosen can be defined in a number of different ways. One may simply select a neighbor from the PI-neighborhood [66], or select a completely random permutation, maybe restricted by some heuristic rules [120]. Good results are reported on an algorithm that switches between these two neighborhoods [117]. Another variant incorporates continuous variables into the simulated annealing [97, 99].

Operational constraints are usually handled by *bounds cooling* or augmentation of the objective function.

In *bounds cooling*, the operational constraints or 'bounds' are relaxed by a value that depends on the annealing temperature. As the annealing temperature decreases, the bounds become more and more tight. Patterns that do not satisfy the (relaxed) constraints at some stage are rejected. In this way, infeasible patterns may be accepted at the beginning of the algorithm, thus widening the search. Towards the end of the search, only (almost) feasible patterns are likely to be accepted.

When using an *augmented objective*, constraints that are not satisfied are penalized in the objective function. In this way, infeasible patterns are only likely to be chosen if their original objective function value is much better than the objective value of the current solution. Towards the end of the search, infeasible patterns have a very low probability of being selected. It is crucial to find good penalty parameter values in order to get a good algorithm, which have to be found by experience.

The evaluation of the trial pattern can in principle be done by an arbitrary core evaluation model, such as a one-and-a-half-group model [117], or a two-group model [120]. The computation time may be decreased by using Generalized Perturbation Theory (GPT, see Section 3.3.1) to evaluate the trial patterns [66, 80, 79]. In [77], a simulated annealing algorithm for the fuel positioning is combined with successive integer linear programming for finding the optimal placement of burnable poisons for each evaluated loading pattern.

3.1.3 Evolutionary algorithms

Evolutionary algorithms (EA) form another class of algorithms that are inspired by the evolutionary principle of survival of the fittest. Starting from a basic population of loading patterns, offspring is created ('generated') either by combining loading patterns (cross-over), or by mutation of single loading patterns in the population. After evaluating the objective function of the offspring patterns, a new population is formed by selecting some members of the current population and some members of the offspring. This –possibly stochastic– selection process depends on the quality of the objective function values of the patterns. Two of the possible stopping criteria are stopping after a pre-determined number of generations, or stopping when no improvement is found during some generations.

Mutation

There is much freedom in how to define combination and mutation operators. One author [5] arranges the fuel elements with respect to their reactivity at BoC. A mutation operation then selects a bundle whose Begin of Cycle reactivity is within a randomly generated range around the BoC reactivity of the current bundle in that position [5]. Another author [25] represents the patterns with bit strings that are combined and mutated. Both methods are combined with expert knowledge (engineering constraints) and heuristics.

Cross-over

The cross-over operation in its simplest form combines a part of one loading pattern, say the inner half of the core, with the other part of another pattern. In practice, lots of refinements are to be made, for example to prevent the placement of the same bundle in both parts [15, 104]. An alternative approach uses no individual bundles during the evolutionary algorithm, but patterns with BoC reactivities are arranged [25]. The advantage is that one does not have to care about double placement of one bundle. Instead, one has to assign bundles that match the BoC reactivity pattern of the 'optimal' solution as well as possible. This approach does not work for equilibrium cycle optimization, and also in other cases it may not be possible to match the BoC reactivity pattern with the available bundles.

Selection

For the selection of the new population there are also many possibilities. If offspring is created from single patterns only (by mutation), one may compare a pattern and its offspring, and select the best of the two. If n patterns together are used to create offspring, one can put these n patterns and all their offspring together, and then select the n best patterns from this set. The selection may also contain a random acceptance probability as in simulated annealing.

As with SA, an arbitrary core evaluation model may be used for objective function evaluation, and the quality of an implementation depends on heuristics and engineering rules that are adopted in the main algorithm. Furthermore, an example exists where expert knowledge is used in creating the offspring [25].

3.1.4 Tabu search

The tabu search (TS) algorithm[1] is designed as a deterministic algorithm that is able to escape from local optimal solutions. Given a neighborhood structure, all neighbors of the current

---

[1] Some authors write 'taboo search'. Although this is more correct English, we use the commonly used term 'tabu search' instead, which is also used by Glover [46], considered as the founding father of the algorithm.

solution are evaluated. The best neighbor is selected as new solution, regardless whether it is better than the current solution or not. This strategy would lead to cycling if no further actions are taken. Therefore a 'tabu list' is maintained. A neighbor or swap that is tabu (*i.e.*, on the tabu list) cannot be selected as new solution, unless it satisfies a certain 'aspiration criterion', which is usually the criterion that it must be better than any previous solution.

The tabu list is considered as the 'memory' of the algorithm. It contains a list of swaps that are performed in previous iterations with a number indicating the 'recency' of the swap. This recency is set to a given value when the swap is executed. After each iteration the recency value is decreased, and when it reaches zero the swap is removed from the tabu list. Different and more sophisticated tabu list implementations are possible. Instead of maintaining a list of swaps, one may for example declare a list of nodes or bundles that have recently been involved in a swap. Also, one may use a long-term type of memory by counting the frequency of certain swaps [47].

We have found one implementation of TS in reload pattern optimization [75]. Here, the neighborhood is restricted by knowledge-based heuristics. The obtained results are not compared to other methods. In [15], it is stated that 'when correctly used the performance is better than SA like algorithms'. This statement is not based on experience in reload pattern optimization, however, while the applicability of this type of statements strongly depends on the problem at hand.

3.1.5 Global to local search

Global to local search [42, 44] is a two phase method. In the first phase, a global search is performed, where random patterns are selected from a large neighborhood. In the second phase, a smaller neighborhood of the best found pattern is searched more thoroughly.

The initial large neighborhood consists of a series of quadruple interchanges. A number of such interchanges is made in sequence, and the corresponding objective functions are just recorded. At the end of a fixed chain length, the best pattern found is selected and a new chain of quadruple interchanges is started from there. If there is no improving solution in the chain, then the local search is started, which is a PI algorithm.

The idea of the algorithm is that, initially, it is important to locate the vicinity of a local optimum. But in order to reach such an optimum exactly, one has to do a more refined local search.

In a more sophisticated variant, the switch from global to local search is not made at once, but stochastic methods are used to make the search gradually more local. The latter implementations can be viewed as variations of SA.

Another variant called *Population Mutation Annealing* (PMA) [41, 39] is an evolutionary algorithm that works with a set of solutions and a mutation operator, where the mutation operator and the criterion whether or not a new pattern is accepted, are stochastic and depend on the annealing temperature.

3.1.6 Other methods in literature

There are a lot of other methods in literature, that all have the property that, in a smart way, a (relatively) restricted number of reload patterns is evaluated, in order to arrive at a local optimal solution.

One of these methods is the use of *neural networks* [60, 61]. With the use of a network with one hidden layer, that is trained with a number of reference loading patterns, the objective function and power peak for any given loading pattern are predicted by the neural network within a reported accuracy of about 2.5% for the power peak and 0.1% for the infinite multiplication factor. Since the evaluation time for one loading pattern is reduced drastically, an optimization method may evaluate much more patterns in the same time, thus (hopefully) increasing the solution quality.

Neural networks can be combined with other methods, *e.g.*, with heuristic rules and fuzzy logic [60]. Axmann [5] uses a neural network to provide starting points for an evolutionary algorithm. A *Monte Carlo method* is applied in [52]. Several other algorithms are described with names as Great Deluge Algorithm, Record to Record Travel and Population-Based Incremental Learning Algorithm are described by [15], but no results of these algorithms on the fuel reloading problem are reported.

An artificial intelligence (AI) based approach, embedded in an interactive system, is used by Galperin et.al. [36, 94]. Based on a set of elimination rules, that forbid certain fuel combinations, and preference rules, that specify likely positions for the different types of bundles, a set of patterns is generated that satisfy these rules. The results of the whole solution set are visualized, and based on these results the user may be satisfied, or modify and refine the set of rules.

A modern trend in combinatorial optimization is the combination of different optimization techniques and heuristics in a so-called hybrid algorithm. Such an algorithm for reload pattern optimization is (under a different name) described in [97], where SA and EA are combined, while the algorithms are monitored and adjusted on the fly by AI techniques.

Since the number of techniques and combinations of techniques is endless, the list of methods reported in this section is certainly not exhaustive. Methods that are not yet mentioned are variational techniques [127] and a Hooke and Jeeves pattern search, combined with engineering rules [56], and more direct search methods may be found in literature that are ever applied to the reload pattern optimization problem. Except for a few techniques that are applied quite regularly, however, most of the methods are only used by one or a few authors.

3.2 Continuous optimization approaches

In direct search methods, the optimization algorithm is more or less decoupled from the core evaluation algorithm. Continuous optimization methods use the model equations of the core evaluation model to guide the search. As explained in the introduction of the current chapter, the reload pattern optimization problem is viewed as a mathematical optimization problem with two types of variables:

1. Variables specifying the fuel assignment (which bundle in which node, in which rotational position, and which BP concentration is included in each node);
2. Variables specifying the core reactivity and depletion.

The optimization problem then is of the following form:

$$\begin{array}{ll} \text{maximize} & \text{the desired objective function;} \\ \text{subject to} & \text{- constraints specifying valid fuel assignments;} \\ & \text{- constraints specifying the physical (neutronics) behavior;} \\ & \text{- operational constraints.} \end{array} \tag{3.1}$$

This general description still leaves a lot of freedom for the exact formulation. Actually, a number of papers have appeared using rather different problem definitions in the form of (3.1). Since a detailed reactor model would lead directly to a complex optimization problem, most studies until now are based on a rather simple core model, usually a nodal optimization model. Since bundles cannot be partially moved to other positions in the core, the resulting models contain binary variables, *i.e.,* variables that only can take the values 0 or 1. In the continuous optimization approaches, this integrality is initially relaxed, and then one obtains an optimization problem that is nonlinear, *i.e.,* the constraints contain nonlinear combinations of variables. The problem is also nonconvex, which means that two solutions may be very good, while solutions in between these two solutions may be worse, or even infeasible. The notions of integrality, nonlinearity and nonconvexity of optimization problems are explained in more detail in the next chapter.

A number of continuous optimization algorithms is applied to solve the problem. We give an overview and a short description of methods that were encountered in literature. The mathematical description of the underlying basic optimization algorithms is postponed to the Sections 4.2 and 4.3.

Most papers use some variant of Sequential Linear Programming (SLP). In this method, the model is linearized around a reference pattern. This linearization is either done in the usual way based on Taylor expansion [67, 122, 131], or with use of generalized perturbation theory (GPT) [29, 62, 63, 77, 78, 82, 113]. Most implementations solve the linearized subproblems within a restricted area around the reference pattern, which comes close to the mathematical optimization idea of a trust region. Just for getting some idea of what this means, one might compare it to the neighborhood in direct search methods, but then in a continuous sense.

A more advanced algorithm that is proposed, combines two linear models to construct a quadratic model [134]. Here, the model is linearized with respect to fuel assignment for given BP concentrations, and linearized with respect to BP concentration for fixed fuel assignment. From these two linearizations a new model is assembled that contains bilinear terms, that is solved a Sequential Quadratic Programming (SQP) method. To our knowledge, this algorithm has not been implemented.

Implementations of the gradient projection method are used to find optimal BP distribution for a fixed pattern [2, 122] and to solve a rather coarse model at once [57]. An odd method using derivatives is an approach based on optimal control theory [127].

The SLP, SQP and gradient projection methods are not aiming to find integer solutions. Thus one may end up with a solution in which fractions of bundles are assigned to different nodes. Various methods are used by different authors to produce integer solutions. In one method, given the assignment variables $x_{i,j}$, a penalty term minimizing

$$\omega_n \sum_{i,j} x_{i,j}^2 (1 - x_{i,j})^2 \tag{3.2}$$

is added to the objective, where ω_n increases during the subsequent SLP iterations [77]. Other implementations use a Branch-and-Bound method in the linear subproblems, either very basic [62, 63], or with more advanced selection and branching rules [67]. In another variant the SLP algorithm only finds a distribution of k_{BoC}^{∞} values in the nodes, and then bundles from the inventory are found that match this distribution as close as possible [29, 122]. This assumes that there is an inventory of bundles available to choose from, which is not the case in equilibrium cycle calculations. In the paper [64] a reload pattern is found using a PI method, and SLP is only used to find an optimal BP distribution.

In the method that was studied in the beginning of our research project, a reduced gradient method is used for the continuous optimization, and a comparison was made between a Branch-and-Bound method and an Outer Approximation method for finding integer solutions [65]. More detailed descriptions of these methods are postponed to the next chapter.

In some papers, continuous optimization is combined with direct search methods. Kim and Kim [63] solve a linearized problem, and use a Branch-and-Bound to find an integer solution to this linear approximation. If no improving pattern is found in this way, the sensitivity coefficients of the LP optimization around the current pattern are used to deduce an improving pairwise interchange. In another implementation [77], an attempt is made to improve the obtained (local optimal) solution of the continuous optimization by applying one hundred random pairwise interchanges and evaluating the corresponding patterns.

Combination the other way around is also possible. Two papers [64, 77] show implementations where the bundles are placed using local search, while in a second phase the Burnable poisons are placed using continuous optimization.

We may conclude that the majority of papers uses some variant of the SLP method, possibly embedded within a trust-region-like method. Most of the implementations use modifications to the standard mathematical optimization methods. Little work is done on investigating the use of other general-purpose nonlinear optimization algorithms.

In this respect, the rather old paper using the gradient projection method [57] is interesting. Despite the fact that the model is much coarser, the model structure somewhat resembles the structure of our model, and the same problems are reported that were also initially encountered in our research, such as the inability of the solver to find initial feasible solutions, and the sensitivity to very small changes in the model parameters. Using current state-of-the-art nonlinear optimization solvers, embedded in a problem-specific framework, we could overcome most of these deficiencies (Chapter 5).

3.3 Tools for general use

This section discusses some existing tools that can be used in the above described optimization algorithms. These tools are meant to speed up the algorithms, either by decreasing the time needed for evaluating the physical parameters of the model, or by steering the optimization process with use of engineering knowledge.

3.3.1 Perturbation theory (PT)

In a local search algorithm, one has to evaluate neighbor loading patterns that are very similar to a loading pattern for which the physical characteristics are already calculated in a previous iterate. Instead of recalculating the new pattern completely, one may try to estimate its physical characteristics from the known neighboring loading pattern.

This technique is known as generalized perturbation theory (GPT), and it has for example been used in an SA framework [66, 79, 80]. GPT is used in the global to local search and the PMA algorithm [41, 43], and also in the linearization step of some NLP-based implementations, to get approximations for the gradients. Computation times of local search algorithms can be greatly reduced when GPT is used to approximately evaluate the neighbors.

In GPT, the effect of a given small perturbation on a desired quantity is approximated. It is based on the theory of adjoint equations, which is closely related to duality theory in nonlinear optimization. For a general description of GPT one is referred to a textbook on operator theory, *e.g.*, [48]. For a description of GPT applied to loading pattern evaluation one may consult one of the papers mentioned above or a text book on nuclear reactor analysis, *e.g.*, [28].

3.3.2 Engineering constraints

Many implementations use one or another form of expert knowledge. One example is the use of expert knowledge to select a starting point [25]. Another example of expert knowledge is the use of engineering constraints. These constraints are used to exclude some fuel configurations that are known to be bad. There are several types of engineering constraints.

Simple geometric heuristics maintain a list of prohibited fuel–node pairs [64, 117, 120]. It may be forbidden, for example, to place fresh fuel on the core boundary or in the core center. Relational heuristics exclude some specific combinations of fuel elements at specific positions in the core. One may for example exclude all patterns with four fresh bundles in a square, or patterns with old bundles paired in inner regions of the core [120]. Similar constraints are used to enforce a checkerboard pattern [121].

Other heuristics exclude the placement of burnable poisons at certain positions [66]. Another class of engineering constraints do not forbid certain positions, but give preference positions for some bundles. This is used to guide a search process when a choice between multiple alternative patterns has to be made [60].

All engineering constraints aim at reducing the search space. In this way, one hopes to find good solutions in less iterations. Care is needed when formulating this type of constraints. It may happen that an engineering constraint excludes the global optimal pattern. Experience and some understanding of the problem structure is required to formulate engineering constraints that are really helpful.

3.4 Conclusion

As has become clear in the previous sections, a lot of optimization methods have been applied to the fuel management optimization problem.

The question raises how all the different methods compare to each other, and how the methods that we are developing will compare to the results from literature. Unfortunately, it is not so easy to answer these questions. There is no standard test set that is used by all authors, each paper presents its own test set, without comparative study with other algorithms on the same test set.

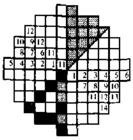
Quite commonly, a test is built up as follows. The test set consists of one core geometry from some arbitrary real-world reactor. With this core geometry, optimization is run on a few fuel inventories that are taken from past cycles of the reactor. The results of the optimization routine are then compared with the loading pattern that was actually implemented in the past. Although this method may be very appropriate to convince the responsible people in these plants of the use of optimization techniques, it does not give any comparison of the different algorithms.

The situation is even worse since there is no standard core model, but in almost any paper, a different core model is implemented. This is partly due to the fact that 'the' model simply does not exist. Using a very detailed model, that is needed for verification of a 'real-world' pattern, simply would take too much time in a local search method and would be too complex in a gradient-based method. Although there is some common notion about which simplifications are justified, there is lot of room to derive many different simplified models, even with different optimization objectives. Another partial reason for the existence of many different models is that different reactor types (LWR, PWR) require different modeling, but this does not explain that there are almost as much different models as there are papers published on reload pattern optimization.

We must conclude that at the current stage, only general remarks about the quality of the different methods can be made. The direct search algorithms usually adopt a black-box method, where the optimization model is separated from the physical model. The level of detail that may be included in the fixed-core evaluation therefore only depends on the available computing time. However, such a method generally converges rather slowly without problem-specific adaptations. The quality and speed of the algorithms is improved by addition of engineering knowledge and use of GPT when exploring the neighborhood. With these additions, some local search methods lead to very satisfactory solutions.

In the gradient based methods the black box is opened. Information from within the black box is used to help the optimization process. This leads to a nonlinear optimization model. The reviewed implementations generally use some variant of SLP to solve this model. In the SLP variants, a linearization around a reference solution is needed, which is obtained by either Taylor expansion or GPT. The validity of the linearization is often restricted to some 'trust region'. All papers use a dedicated implementation, where the linearization is implemented by the authors. The LP subproblems are solved by standard routines. Since large systems of equations are to be solved in the linearization step, or large LP problems are obtained, the used models are generally small.

The literature review could give the feeling that SLP is the preferred NLP method (since most authors use it), and that even that method is only applicable to smaller models, so that local search is still the only choice to solve the reload pattern optimization problem. Other, more powerful NLP algorithms are developed, however, that perform better in general than SLP. Moreover, excellent implementations of these algorithms are available, that can handle much larger and more difficult optimization problems. In the next chapter, we turn our attention towards such algorithms, and their applicability to the fuel management optimization problem.



Chapter 4

Mixed-integer nonlinear optimization

In Chapter 2, we modeled the fuel reload optimization problem. After a number of simplifications we finally arrived in Section 2.3 at an optimization model containing both nonlinear constraints and integer variables. This means that, in terms of mathematical optimization, the model can be classified as a mixed-integer nonlinear optimization problem (MINLP-problem)[1]. In this chapter, we will discuss general properties of MINLP models, and review possible techniques for solving these problems. This discussion will be quite general, although sometimes links to the fuel management problem are made. Algorithmic and implementation issues that are specific to the fuel management problem are postponed until Chapter 5.

Section 4.1 starts with a discussion of the general structure of MINLP problems. Most practical algorithms break down into a nonlinear optimization part and an integer optimization part. Issues related to nonlinear optimization are discussed in more detail in Section 4.2, while issues related to integer optimization will be treated in Section 4.3. The algorithms described in these sections are only able to guarantee local optimality of solutions. Methods that aim at finding global optimal solutions are described in Section 4.4.

4.1 General structure of MINLP

The basic model in Mathematical Optimization is the linear optimization (LP) model. In linear optimization, all variables are defined over connected intervals, and both the objective function and all constraints are linear. For LP there exist efficient solution methods [31]. Linear simplex methods [85, 89] are not proven to be efficient in theory, but are very efficient in practice. Interior point methods for LP [110] are efficient in theory and very efficient in practice, especially for large problems. LP problems with tens of thousands of constraints and variables can be solved towards a global optimum with today's LP solvers on a PC.

A MINLP model is an extension of an LP model in two ways: firstly the objective function

---

[1] Standard classification of mathematical optimization models uses abbreviations ending with 'P', denoting 'Programming' (even Mathematical Optimization is commonly stated as Mathematical Programming, MP). Although we adopt this standard in this thesis, we often use the word 'Optimization' instead of 'Programming', since this avoids confusion with 'programming' in the sense of writing Fortran or C programs, and describes much better what we are talking about.

and/or some or all of the constraints are nonlinear, and secondly some or all of the variables have to be integer. Either of these extensions separately can be enough to make finding a global optimal solution a very difficult, frequently impossible task.

Regarding integrality, it is illustrated by Figure 4.1 that simple rounding of a continuous optimum is generally a bad method. It also illustrates that the discrete optimum may be far away from the continuous optimum.

Nonlinearity may introduce numerical difficulties, that make it difficult to converge to even a local solution. It may also introduce nonconvexity. A function is *convex* if any two points on the function can be connected by a straight line that lies entirely on or above the function itself. More precisely, a function f is convex if for any two points x_1 and x_2 it holds that

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2).$$

In Figure 4.2a, for example, a one-dimensional function is shown, that is clearly nonconvex: if we draw a straight line between the two end points of the function, large parts of the function lie above this line. From this figure it becomes clear that nonconvexity of the objective function may introduce *local optimal solutions:* solutions that are optimal in a small region around the solution, but not necessarily optimal over the whole feasible area. A local optimal solution that has the best objective value out of all local optimal solutions over the whole feasible area is called a *global optimal solution.* Finding an arbitrary local optimal solution is generally not a very hard task, but ensuring that it is a good local optimum, or even the global optimum, can be a hard and practically impossible task. Graphically, in Figure 4.2a it is not too difficult to find the bottom of just some valley, starting from any point. Guaranteeing whether this is the global deepest valley, or even that it its level is only within a given distance from the deepest level, is a much harder task.

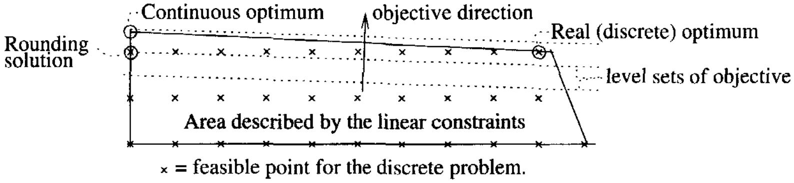
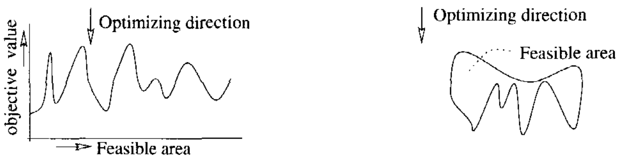


FIGURE 4.1: Optimization of a discrete problem.

Besides nonconvexity of the objective function, also the feasible region may be nonconvex, which makes the problem hard even if the objective value is linear. A feasible region is *convex* if any straight line connecting two points in the feasible region, lies itself entirely in the feasible region. In Figure 4.2b a nonconvex feasible region is given. The objective is linear: find the most southern point in the feasible area, but due to the nonconvexity of the feasible region, it is again difficult to find a global optimal solution; the circled point, for instance, is a local optimal solution, but from a local view, it is not clear at all whether this is the globally optimal solution, or only a bad local optimum. In this two-dimensional example, one may immediately conclude that there is a better solution, but in more-dimensional cases, where visualization is impossible,



a) One-dimensional example of local optimality. Nonconvex feasible area, nonconvex objective. Find the deepest valley in a foggy environment.

b) Two-dimensional, nonconvex feasible area. The objective function improves towards the page bottom. Find the most southern point in a foggy environment.

FIGURE 4.2: Optimization of two different nonconvex problems.

this is a much harder, and frequently impossible task. The feasible area in this example is still connected, but even this need not to be the case in general.

The loading pattern optimization models as derived in Section 2.3 contain a number of nonconvex constraints. Furthermore the assignment variables are integer. From the examples above, it will be clear that this combination of difficulties leads to a problem that is really hard to solve. Indeed it is so hard that there are no algorithms known in literature that guarantee global optimal solutions to the loading pattern optimization problem in any reasonable time. Section 4.4 will deal with algorithms that aim at finding global optimal solutions, or at least a quality bound on the obtained solutions. These methods may only be expected to work for rough models and small core sizes.

In the next section we ignore integrality constraints and concentrate on the problem of solving the nonlinear optimization (NLP) part. In Section 4.3 the integer programming part is discussed.

It may be good to mention that we do not discuss multi-objective optimization, although some work on this approach is done in fuel management optimization literature [100]. Good references for this specialized topic are for example [95, 119, 133]

4.2 The nonlinear part

In this section, we focus on optimization problems of the following form:

$$\begin{aligned} \min \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \cdots, m, \\ & h_j(x) = 0, \quad j = 1, \cdots, p. \end{aligned} \tag{4.1}$$

The functions $f(x)$, $g_i(x)$ and $h_j(x)$ may be linear or nonlinear. Both the objective function and the feasible area may be nonconvex. The variables are the entries of the vector x and these may attain any real value. As we have already seen in Figure 4.2, such problems may have many local optima, and it may be very difficult or impossible to find a global optimum. In this section we only consider algorithms for finding local optimal solutions. Most methods designed to solve such problems can be categorized as follows.

- Derivative-free direct search methods, also known as Black box methods. The best known of these methods is the nonlinear simplex method of Nelder and Mead [69, 129]. As methods like Genetic Algorithms and Simulated Annealing, they are essentially black box methods, where only function values are needed. They are essentially methods for unconstrained optimization, although they may be adapted in some extent to solve constrained problems, for example by penalizing constraint violations in the objective. These methods are quite widely used in engineering because of their simplicity and because of their applicability when no derivative information is known. Although they are also frequently used when derivative information is obtainable, derivative-based methods perform usually better in this case.

  More advanced versions of direct search methods use an approximate model to improve the search [20, 21]. Based on function evaluations in a number of points in the search space, a quadratic model is built. This model is minimized using the trust-region method, and in the optimal point of the approximating model, a new function evaluation in the original problem is performed. This function evaluation is used to refine the approximating model, until a satisfactory solution is obtained. Another sophisticated variant is the space-mapping method [6], where a simplified model is used to guide the search for an optimal solution of a detailed model. A very general overview of direct search algorithms is given by Powell [105].

- Derivative-based optimization algorithms. This is often the category of choice when derivative information can easily be obtained. First order, and possibly second order derivative information is obtained in each iteration point, and is used to find improving search directions. There are basically three subclasses.

  1. Gradient (or first order derivative) methods only use first order derivative information.
  2. Hessian (or second order derivative) methods use both first and second order derivative information.
  3. Quasi-Newton methods use first and second order derivative information. The second order derivatives are not computed explicitly in each iteration, but they are approximated by using the first order information from previous iterations.

  There exist many derivative based optimization algorithms, and although there is some experience about which methods are the better ones, the efficiency and robustness of the different algorithms may depend on the specific problem at hand. Since this is a very important class of algorithms in our study, it will be discussed in more detail in the sequel.

- Other approaches. Several less frequently used algorithms exist, including methods based on duality (primal-dual methods). Using model-specific relaxations, decompositions or branching strategies, it is sometimes possible to obtain solutions that are very difficult to find with other optimization algorithms. Such an approach often requires more effort and more insight from the modeler, but may lead to substantially better solutions or shorter running times.

Before discussing gradient-based approaches, we first introduce some basic theoretical notions for nonlinear optimization. This description is very brief. For a more complete introduction to the theory of linear and nonlinear optimization, one may consult some textbook on this topic [9, 32, 45, 88].

4.2.1 Basics of nonlinear optimization theory

Suppose we have the optimization problem (4.1):

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \cdots, m, \\ & h_j(x) = 0, \quad j = 1, \cdots, p. \end{aligned} \tag{4.2}$$

In words, the objective of the optimization is to find such a solution x^* that has the lowest objective function value, out of all possible solutions x that satisfy the constraints.

Associated with the optimization problem (4.2) is the *Lagrange function* $L(x,u,w)$:

$$L(x,u,w) := f(x) + \sum_{i=1}^{m} u_i g_i(x) + \sum_{j=1}^{p} w_j h_j(x) \tag{4.3}$$

(also known as Lagrangian), which is defined for positive u_i, $i = 1, \cdots, m$. The elements of u and w are known as the *Lagrange multipliers*.

In the sequel, we assume that the objective and constraint functions are continuously differentiable functions. Then, the theorem of Karush, Kuhn and Tucker gives first order necessary conditions for a point x to be a local minimizer.

Theorem 4.1 (KKT-conditions) *Given the optimization problem (4.2), where f, g_i, $i = 1, \cdots, m$ and h_j, $j = 1, \cdots, p$ are differentiable. Let $\bar{x}$ be a point satisfying all constraints, and let $\nabla g_i(\bar{x})$, $i = 1, \cdots, m$ and $\nabla h_j(\bar{x})$, $j = 1, \cdots, p$ be linearly independent at $\bar{x}$. If $\bar{x}$ is a local optimum of (4.2) then there exist scalars $\bar{u}_i$, $i = 1, \cdots, m$ and $\bar{w}_j$, $j = 1, \cdots, p$ such that*

$$\begin{aligned} \nabla f(\bar{x}) + \sum_{i=1}^{m} \bar{u}_i \nabla g_i(\bar{x}) + \sum_{j=1}^{p} \bar{w}_j \nabla h_j(\bar{x}) &= 0, \\ \bar{u}_i g_i(\bar{x}) = 0, \quad & i = 1, \cdots, m, \\ \bar{u}_i \geq 0, \quad & i = 1, \cdots, m. \end{aligned} \tag{4.4}$$

Note: The first equation of (4.4) is equal to the equation $\nabla_x L(\bar{x}, \bar{u}, \bar{w}) = 0$; the scalars $\bar{u}_i$ and $\bar{w}_j$ from the theorem are just the Lagrange multipliers. A point $(\bar{x}, \bar{u}, \bar{w})$ satisfying (4.4) is called a KKT point.

For a proof of Theorem 4.1, the reader is referred to *e.g.*, [9]. A graphical illustration of the theorem is given in Figure 4.3. In the left picture, the feasible area is given for the case that no equality constraints are present. Suppose x^1 in the interior of the feasible set is the optimal solution, then $\nabla f(x^1)$ must be zero, otherwise the direction $-\nabla f(x^1)$ would be an improving direction. Indeed, since $g_i(x^1) < 0,\ i = 1, \cdots, m$, it holds that $u_i = 0,\ i = 1, \cdots, m$, and hence the KKT conditions in x^1 reduce to $\nabla f(x) = 0$.

If, on the other hand, a boundary point such as x^2 is the optimal solution, the gradient $\nabla f(x^2)$ may be nonzero, but for any direction s inwards the feasible region, it must hold that the directional directive $\nabla f(x^2)^T s \geq 0$. This is true if and only if $\nabla f(x^2)$ is a negative multiple of $\nabla g_2(x^2)$. So for some nonnegative (possibly zero) u_2 it must hold that

$$\nabla f(x^2) = -u_2 \nabla g_2(x^2)$$

which is exactly what is posed in (4.4).

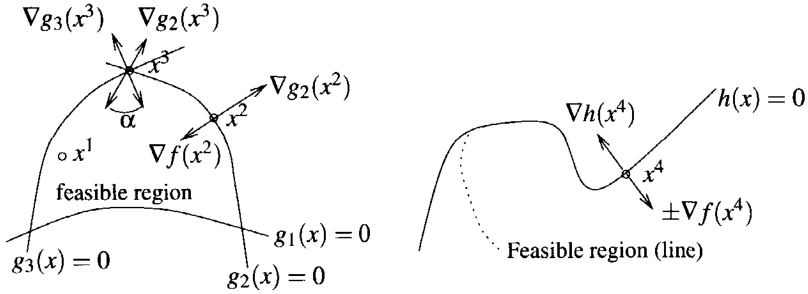


FIGURE 4.3: Illustration of the KKT conditions.

In the case x^3 is the optimal solution, the gradient of f may vary in the cone α, and this is modeled in (4.4) since both u_2 and u_3 may be nonzero (because $g_2(x^3) = g_3(x^3) = 0$).

The right hand side picture of Figure 4.3 shows a case with a single equality constraint. In this case it is sufficient that the gradient of the objective function is tangential to the gradient of the constraint, either in positive or negative direction. This is in accordance with Theorem 4.1, where no sign-restriction is posed on the w-variables.

In Theorem 4.1 the condition is posed that constraint gradients are linearly independent. If this condition is not satisfied, a point may be a local optimum without satisfying the KKT conditions. A standard counterexample for this, as is given by Kuhn and Tucker, can be found in many textbooks on nonlinear optimization, *e.g.*, [9, p.147].

In the case that there are no equality constraints, and the functions $g_i(x)$ are concave, the condition of linear independence of constraint gradients is equivalent to the condition that

$$\text{a point } \tilde{x} \text{ exists such that } g_i(\tilde{x}) < 0,\ i = 1, \cdots, m. \tag{4.5}$$

This restriction is known as the *Slater regularity condition.*

The KKT conditions are closely related to the Lagrange function. It can be proven that any KKT point $(\bar{x}, \bar{u}, \bar{w})$ is a stationary point of the Lagrange function.

Second order conditions

For a point to be a local minimum, it is necessary that it satisfies the KKT conditions. This is not sufficient, however. The KKT conditions also hold in other stationary points, such as maxima, saddle-points (Figure 4.4) and inflection points. So in order to determine whether a KKT point is a local minimum, one needs more information. The KKT conditions can be improved by using second derivatives, describing the curvature of the problem functions at KKT points. In one dimension, the sign of the curvature (positive, zero, negative) corresponds to the sign of the second derivative. In more dimensions, the sign of the curvature in direction s at position x corresponds to the sign of

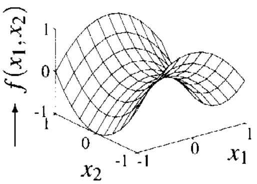


FIGURE 4.4: $f(x_1,x_2) = x_1^2 - x_2^2$. Saddle point at (0,0): $\nabla f(0,0) = 0$.

$$s^T H(x)s,$$

where $H(x)$ is the second derivative matrix (*Hessian*) of the function f at x. If the curvature is positive in any nonzero direction, then the Hessian is said to be positive definite (PD). If the curvature is nonnegative in any direction, then the Hessian is positive semi-definite (PSD).

Consider a KKT point in the interior of the feasible area, so there are no constraints active in this point. A *necessary* condition to be a local minimum is that the objective function in that point has a *nonnegative* curvature in any direction. On the other hand, a *sufficient* condition for the KKT point to be a local minimum is that it has a strictly *positive* curvature in any direction.

In the case that the curvature is nonnegative in any direction, but zero in one or more directions, second order information is still not sufficient to identify a local minimum. Compare for example the point $x = 0$ for the two functions $f_1(x) = x^3$ and $f_2(x) = -x^3$.

Summarizing:

- x is a local minimum $\Rightarrow$ KKT conditions hold at x and $H(x)$ is PSD;
- KKT conditions hold at x and $H(x)$ is PD $\Rightarrow$ x is a local minimum.

In case there are active constraints in the KKT point, one has to consider the Hessian of the Lagrange function instead of the Hessian of the objective function. This Hessian has to be positive (semi-)definite for directions y inwards the feasible area only. Checking this condition requires in general a copositivity check (see also Section 7.4), which is in itself a very hard problem.

Let us further point out that most optimization packages do not calculate the second derivative of the Lagrangian explicitly. For all these reasons, the practical use of the second order constraint qualifications is very limited.

It may be concluded that it is practically impossible to formulate general criteria to check whether even a local minimum is reached. For some problems this may imply that solvers may detect that the KKT conditions are satisfied, and report a minimal solution, while the point is in fact a saddle point or even a maximum. A striking illustration is given by the unconstrained minimization problems

$$\text{minimize} -x_1^2 + x_2^2$$

(the problem of Figure 4.4), and even worse, the problem

$$\text{minimize} -x_1^2 - x_2^2.$$

Current state-of-the-art solvers that are started from the initial point (0,0), terminate immediately with the message that the current point is optimal. In practical models, this situation is easily circumvented by using appropriate starting points. Also, there exist techniques to avoid reaching saddle points in cases as in Figure 4.4. A more severe problem in practical implementations is that only local minima are reached. Only global optimization techniques can give a qualification on how far the current local minimum objective value is away from the global optimal value.

In the next three sections we concentrate on solution methods of (4.1) that use first order and second order information. These methods are usually descent algorithms, in which the objective value is improved at each iteration. The description starts with unconstrained optimization algorithms. These methods belong to the building blocks of constrained optimization techniques. After a discussion on how to obtain derivative information in Section 4.2.3, attention is paid to constraint optimization in Section 4.2.4. The list of algorithms is by far not complete, and only algorithms that are implemented in one of the solvers that were used in our research are discussed in more detail.

4.2.2 Unconstrained optimization methods

The general structure of most unconstrained minimization algorithms is either line search based, or trust-region based. Both types of algorithms are discussed below.

Line search based methods.

Let x^0 be the starting point. In a step length based method, the kth iteration is comprised of the following steps.

1. *Convergence test.* If the convergence conditions are satisfied, stop with solution x^k.
2. *Search direction.*Calculate a nonzero vector s^k representing the search direction.
3. *Line search.* Calculate a certain positive step length α^k, such that $f(x^k + \alpha^k s^k) < f(x^k)$.

4. *Update the current point.* Let $x^{k+1} = x^k + \alpha^k s^k$, update $k = k + 1$, and return to step 1.

In summary, the main ingredients are the convergence test, the search direction calculation, and the line search.

The line search is in fact a new minimization problem in just one dimension. This can be either an exact minimization, or the search can be stopped earlier. There is some trade-off between the effort spent in the line search and the resulting benefits for the overall optimization.

The basic *first derivative based* unconstrained minimization technique is the *steepest descent algorithm*. Consider the Taylor expansion of f at x^k in a (yet unknown) direction s^k:

$$f_{(T1)}(x^k + s^k) = f(x^k) + \nabla f(x^k)^T s^k.$$

The steepest descent method now searches a direction s^k such that $f_{(T1)}(x^k + s^k)$ is minimized, assuming that s^k has a normalized length, $||s^k|| = ||\nabla f(x^k)||$. This is equivalent to minimizing the term $\nabla f(x^k)^T s^k$, which leads to the search direction

$$s^k = -\nabla f(x^k).$$

Before performing the actual step, a step length is computed. If the step length algorithm is such that it ensures sufficient decrease of the objective function, convergence of the algorithm to a stationary point can be proven. It is generally not an efficient method; often, even for convex functions, many iterations are needed to reach the optimum. More elaborate first derivative based methods are for example conjugate direction methods, that are not further discussed here.

Second derivative based methods use second order Taylor expansion:

$$f_{(T2)}(x^k + s^k) = f(x^k) + \nabla f(x^k)^T s^k + {}^1\!/_2 (s^k)^T \nabla^2 f(x^k) s^k.$$

A search direction s^k is selected that minimizes $f_{(T2)}(x^k + s^k)$, which is equivalent to minimizing

$$\nabla f(x^k)^T s^k + {}^1\!/_2 (s^k)^T \nabla^2 f(x^k) s^k. \tag{4.6}$$

By setting the gradient of (4.6) with respect to s^k to zero, one obtains that the search direction can be computed by solving the system

$$\nabla^2 f(x^k) s^k = -\nabla f(x^k). \tag{4.7}$$

The resulting method is known as *Newton's method*, and the direction s^k is the *Newton direction*. Newton's method provides very fast convergence if the current point x^k is 'close' to the optimum, when the objective function is very well approximated by the quadratic model $f_{(T2)}(x^k + s^k)$, and if the Hessian is positive definite (PD). A convex quadratic function is minimized in one iteration, provided that exact first and second derivatives are used. If the Hessian is not PD, then the solution s^k of (4.7) may be an uphill direction (in which case $-s^k$ can be selected as a downhill direction) or it may be impossible to solve (4.7) when the Hessian is singular. In this case one may fall back to the steepest descent direction. A well-known modified Newton method that deals in an effective way with Hessian matrices that are not PD, is described by Fiacco and McCormick [32].

Trust-region methods.

Trust-region methods are different from line search based methods. In Trust-region methods, we start again with the (second order) Taylor expansion around x^k:

$$f_{(T2)}(x^k+s^k) = f(x^k) + \nabla f(x^k)^T s^k + {}^1\!/_2 (s^k)^T \nabla^2 f(x^k) s^k \tag{4.8}$$

and an s^k is searched that minimizes

$$\nabla f(x^k)^T s^k + {}^1\!/_2 (s^k)^T \nabla^2 f(x^k) s^k. \tag{4.9}$$

When in line search based methods a minimizing direction s^k of (4.9) was found, the method proceeded with a line search. In Trust-region methods, (4.9) is minimized without line search, but under the restriction that

$$||s^k|| \leq \Delta$$

for some positive Δ. So, the quadratic model is optimized within a ball (if the 2-norm is used) or a box (if the ∞-norm is used) around the current solution. For the 2-norm, this is equivalent to the unconstrained minimization of

$$\nabla f(x^k)^T s^k + {}^1\!/_2 (s^k)^T (\nabla^2 f(x^k) + \lambda I) s^k$$

where λ is related to Δ. The case $\lambda = 0$ corresponds to $\Delta \to \infty$, and then s^k will become the Newton direction. If $\lambda \to \infty$, this corresponds to $\Delta \to 0$, and the direction of s^k will approach the steepest descent direction (although $||s^k|| \to 0$).

The process is controlled by comparing the 'predicted' reduction in the objective function $f_{(T2)}(x^k+s^k)$ and the actual reduction in the real objective function $f(x^k+s^k)$. If the prediction was very good, then the trust region is expanded in the next iteration; if the prediction was very bad, then the current iteration is repeated with a smaller trust region.

4.2.3 Obtaining derivative information

Except for the direct search methods, all optimization algorithms require first derivative information, or even second derivative information. First order derivative information can be supplied

- explicitly by manually programming the derivatives. The disadvantage is that it can be a tedious task to compute the first derivatives, and errors are coming in very easily.
- by an automatic (symbolic) differentiation tool. There are several of these tools available, that can differentiate quite complex functions. The symbolic derivative is computed once, and then it is evaluated each time a derivative is needed. Optimization modeling languages usually supply these tools.
- numerically by finite difference techniques. Given a point x^k, the approximate derivative in the direction of the (normalized) vector s is computed as

$$\frac{f(x^k+\delta s) - f(x^k)}{\delta}$$

> for some sufficiently small δ. The advantage of this method is that only function evaluations are needed. Two disadvantages of this method are the lower accuracy, and the longer computation time. In general it is better to spend some effort in supplying exact derivatives.

Regarding second derivatives, it is even more difficult and error-prone to supply them manually. As an alternative, they can also be supplied by automatic differentiation tools or finite difference approximation. Another practical alternative is given by Quasi-Newton methods. In these methods, the Hessian is not completely recomputed at each iteration, but it is approximated using the first derivative information from subsequent iteration points. Every time a new gradient is computed, it is used to update the approximated Hessian. This technique assumes that the Hessian does not change too much during subsequent iterations, an assumption that is certainly true for quadratic and bilinear functions. Most state of the art nonlinear optimization solvers use a Quasi-Newton scheme to approximate the Hessian.

4.2.4 Constrained optimization methods.

The presence of constraints in the problem has major consequences for the algorithms. Instead of searching for a point x where

$$\nabla f(x) = 0,$$

one now has to find a point that satisfies the KKT conditions (4.4). There is a major difference in problems with only linear constraints and problems with nonlinear constraints. Many algorithms to solve the latter create subproblems with linear constraints, that are solved with dedicated algorithms. In our study we used different solvers that make use of various algorithms. None of the algorithms is better than the others in an absolute sense; the efficiency and robustness of the algorithm depends strongly on the specific type of problem, and even on specific implementation details of the algorithm. A brief overview of the algorithms is given below. The algorithmic ideas are illustrated in more detail for the Reduced Gradient method (RGM) and the Generalized Reduced Gradient (GRG) method. These methods were chosen because several of the software packages that we used in our research, invoke RGM as a suboptimization algorithm, and the –in our research– most successful package (CONOPT) is based on the GRG method. A more detailed description of all listed algorithms is found in Appendix A and in the literature [9, 45].

Sequential or Successive Linear Programming (SLP)

This method, in some earlier literature also called Approximation Programming [29, 49], is, in several variants, frequently used in fuel management literature. In each iteration, the objective and the constraints are linearized around the current point. The resulting linear problem is solved towards optimality within a specified region around the current point. At the new point, a new linearization is made, until convergence is reached.

The linearization is usually only valid on a very local scale. Therefore, constraints may be added to the linearized problem that limit the maximum deviation from the current point. Another possibility is to use the linear problem only to generate a search direction, followed by a line search to compute a step length.

SLP is used as 'sub'-algorithm in some of the state-of-the-art nonlinear solvers, but it is considered as a method that is not so effective to solve general nonlinear optimization problems.

Reduced Gradient Method (RGM)

This algorithm solves problems with a nonlinear objective function, and linear constraints. We assume that, except nonnegativity of the variables, only equality constraints are involved:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & Ax = b \\ & x \geq 0. \end{aligned}$$

The method starts –if needed– with calling a linear solver to find an initial solution that satisfies all the constraints. Then, the gradient (and possibly the Hessian) of the objective function are computed over the null-space of A, *i.e.*, over those directions in which the solution stays feasible. Given n variables and m linear independent constraints, this means the computation of a search direction in a 'reduced' space with only $n-m$ nonnegative variables. The gradient in this null space is called the *reduced gradient*. After finding a search direction, the values of the remaining m variables are uniquely determined from the $n-m$ variables in the reduced space, giving a search direction in the full search space. Finally, a line search is performed in this direction[2].

The ideas of the method are illustrated in the left picture of Figure 4.5. For the purpose of graphical illustration, a problem is shown with linear inequality constraints $a_i^T x \leq b_i$, $i = 1,2,3$. In an interior point like x^1, the steepest descent direction –or possibly a Quasi-Newton direction– is choosen. In this search direction, say s^1, a line search is performed, such that the new point stays feasible. So, if the objective function is monotonically decreasing in this search direction s^1, an exact line search will lead to the next iteration point x^2. In this point, a search direction is computed in the reduced search space consisting of all directions s for which $a_3^T s \leq 0$. Graphically, the set of possible search directions is given by the cone α^2. The resulting search direction can be any direction in this cone, including the two directions tangential to the constraint. Suppose that the resulting search direction is s^2, then a line search is performed in that direction. The minimum in this direction may be anywhere between x^2 and the opposite boundary; for illustration we assume here that it is just x^3, that happens to be on the intersection of two constraints. This is a nice point to illustrate the computational ideas behind the algorithm. In the algorithm, the constraints are converted to equality constraints

$$a_i^T x + z_i = b_i, \quad i = 1,2,3. \tag{4.10}$$

In x^3, $z_1 = z_2 = 0$. Furthermore, it holds that, given an arbitrary nonnegative value of z_1 and z_2, the variables x_1, x_2 and z_3 are uniquely determined by solving the system of equations (4.10):

$$\begin{pmatrix} x_1 \\ x_2 \\ z_3 \end{pmatrix} = B^{-1} \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \tag{4.11}$$

---

[2] for a more precise description in terms of basic and non-basic variables, see Appendix A.

for some matrix[3] B. So, it is sufficient to compute a nonnegative improving direction in the space determined by z_1 and z_2. Due to the nonnegativity of z_1 and z_2, such a direction will be in the cone α^3. The objective gradient in the space determined by z_1 and z_2 is obtained by transforming the original objective gradient in the (x_1,x_2)-space to the reduced gradient in the (z_1,z_2)-space using (4.11). The search direction in the nonnegative quadrant is found using steepest descent or a Quasi-Newton based method. The search direction in the (z_1,z_2)-space is transformed back to the (x_1,x_2,z_1,z_2,z_3)-space, and a line search is performed in this direction. In this line search, we stay feasible with respect to constraint a_3 by requiring that z_3 is nonnegative.

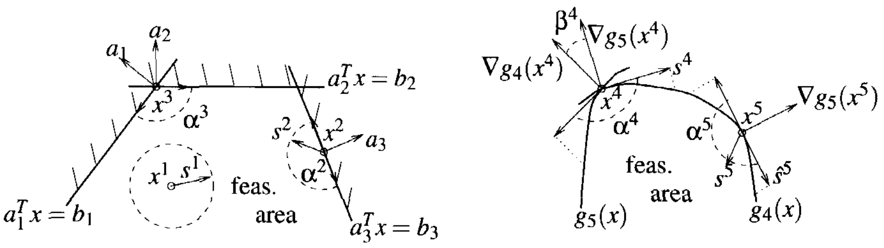


FIGURE 4.5: Illustration of the RGM (left) and the GRG method (right).

Sequential Quadratic Programming (SQP)

In each iteration of an SQP method, the second order Taylor expansion of the Lagrangian is minimized, subject to linearized constraints. This minimization can be performed by a dedicated Quadratic optimization (QP) routine, or by the RGM. Since the Lagrangian is used, estimates for the multipliers u_i and w_j have to be provided initially, and they are updated in every step.

Augmented Lagrangian Method (ALM)

This method resembles the SQP method. However, instead of optimizing a quadratic approximation of the Lagrangian in each reduced gradient iteration, it minimizes a slightly modified version of the Lagrangian (known as the *augmented Lagrangian*), with a given set of Lagrange multipliers, subject to linearized constraints. The Lagrange multipliers are updated in each iteration.

Augmented Lagrangian Trust Region method (ALTR)

As in the ALM, the augmented Lagrangian is minimized in each iteration. Instead of minimizing subject to linearized constraints, this augmented Lagrangian is now minimized with a Trust Region method, possibly with bound constraints on the variables.

---

[3] The matrix B is a *basis* matrix, as is explained in Appendix A.

Generalized Reduced Gradient (GRG)

In SQP and ALM, a linearization of the constraints is made, and then the subproblem with linearized constraints is solved to optimality by a separate optimization routine. In the GRG method, there is no such distinction in an 'inner' and an 'outer' optimization problem, but the linearization is updated in each RGM iteration. Moreover, the line search to compute the step length is more sophisticated. During the line search, the search direction may be adapted in order to find a point that still satisfies the nonlinear constraints. These two modifications ensure that constraint gradient and objective function evaluations are only performed in feasible points.

In order to describe the GRG method in more detail, consider Figure 4.5. As the RGM, the GRG method is devised for equality constraints, but the picture illustrates inequality constraints. If we have only linear constraints as in the left picture of Figure 4.5, the GRG method reduces to the RGM. In case of nonlinear constraints, as in the right picture, the algorithm becomes a bit more complicated. In the example there are two inequality constraints $g_i(x) \leq 0$, $i = 4,5$. Both constraints are active in the current optimization point x^4. In order to find a reduced gradient search direction, the constraints are linearized around x^4 and a search direction s is computed such that $\nabla g_4(x^4)^T s \geq 0$ and $\nabla g_5(x^4)^T s \geq 0$, corresponding to a search direction within the cone α^4. This search direction is found using a reduced gradient in a reduced search space, similar as described in the paragraph dealing with the RGM. Contrary to the linear case however, it may happen that the resulting search direction points outward the feasible set, as illustrated by direction s^4. In that case the search direction has to be repaired in order to maintain feasibility. To this end, the standard line search that finds a point

$$x^{k+1} = x^k + \lambda s^k$$

for some $\lambda > 0$, is extended with a correction term, such that

$$x^{k+1} = x^k + \lambda s^k + p^k(\lambda), \tag{4.12}$$

where $-p^k(\lambda)$ is in the cone β^4 spanned by $\nabla g_4(x^4)$ and $\nabla g_5(x^4)$. The vector $p^k(\lambda)$ is computed by iterative methods as follows. Consider the set of constraints, that can be written down as

$$g_i(x) = 0, \;\; i = 1, \cdots, m.$$

Note that we assume that all constraints are converted to equality constraints here. The point x^k satisfies these constraints. For fixed s^k and λ, the expression (4.12) for x^{k+1} is filled in into the system of constraints, resulting in a system of nonlinear equalities in $p^k(\lambda)$. The variables $p^k(\lambda)$ are then found by iterative methods, *e.g.*, by a Newton-Raphson method. Note that it is not guaranteed that $f(x^k + \lambda s^k + p^k(\lambda)) < f(x^k)$, but assuming that the constraints are smooth, $p^k(\lambda)$ approaches zero for sufficiently small λ and the search direction approaches the decreasing direction s^k.

Inequality constraints can be handled in two ways. The easiest way is to transform them into equalities by addition of slack variables. The other way is the use of an active-set strategy. If one or more inequalities are active at the current solution, and some of these inequalities are violated during the line search, the correction step $p^k(\lambda)$ will be such that a subset of these violated inequalities is active in the resulting solution. This subset, called the active set, can be treated as equalities in retaining feasibility. The difficulty here lies in the determination of the right active set, since it must be chosen such that in the new solution, the inequalities not in the active set are also satisfied. Therefore, a change of the active set may be necessary during the line search.

Summarizing, the GRG method requires more computational effort per iteration than one 'inner' iteration of SQP and ALM, but the overall number of iterations generally decreases, especially for strongly nonlinear problems. In such a case, SQP and ALM need extra effort at each 'outer' iteration since the solution of the linearized subproblem may be quite far away from the feasible set defined by the nonlinear constraints. On the other hand, the steps in a GRG method can become very small, because it may be difficult to maintain feasibility during the line search. An advantage of GRG methods, and also of the Interior Point Methods discussed below, is that objective function evaluations are only performed in feasible points. In many applications, the objective function is undefined or gives unreliable values outside the feasible region. In ALM for example, it may happen that in a subproblem the objective is evaluated outside the feasible region, which makes this method less suited in such cases.

Interior Point Methods (IPM)

Suppose we have only the inequality constraints $g_i(x) \leq 0$ and no equality constraints. The constraints can be replaced by a so-called *barrier function* in the objective:

$$\text{minimize } \frac{f(x)}{\mu} - \sum_{i=1}^{m} \ln(-g_i(x)), \tag{4.13}$$

where $\mu > 0$. The function $\ln(\cdot)$ has the property that it is convex and that both the function and its first derivative go to infinity when the argument goes to zero. Hence, when the initial point x is feasible, any descent method will find a feasible minimizing solution. In an interior point method, the function (4.13) is minimized using a Newton's method, and the parameter μ is gradually decreased during subsequent iterations.

Combination of methods

A commercial available general nonlinear optimization solver usually consists of a combination of various methods. The GRG solver CONOPT [27], for example, uses two variants of the steepest descent algorithm to find the search direction when the model seems to be almost linear. Otherwise it uses a Quasi-Newton method. If the model continues to be almost linear for a number of iterations, it uses an SLP subroutine to generate a search direction for the GRG method. Also, different line search strategies can be chosen, based on the model properties.

4.3 The integer part

After obtaining an optimal solution of the continuous relaxation, one has to find an integer solution. As was already illustrated by Figure 4.1 for the linear case, this is generally not a trivial task. Simple rounding of the continuous solution may give a solution that is far from optimal, or that is not even feasible. Complete enumeration of all possible integer assignments is out of question. Suppose there are n binary variables. Then there exist 2^n possible integer solutions. This number can only be enumerated for very small problem sizes. It may be concluded that more robust techniques are needed in order to find reliable integer solutions.

One solution method that at first sight seems to eliminate the problem in the nonlinear case is to replace the integrality constraints by nonlinear constraints. In case we have general integer variables, the integrality restriction is equal to the nonlinear constraint

$$\sin(\pi x) = 0.$$

In case we have 0-1 variables, the integer restriction is equivalent to the constraint

$$x(1-x) = 0. \tag{4.14}$$

One can easily see however, that this reformulation only enlarges the difficulties, since the area of feasible solutions becomes highly disconnected. Sometimes, the latter transformation (4.14) is used as a concave penalty term in the objective function. Generally, this leads to a model where many of the integer solutions are local minima. When a solution method reaches the vicinity of such a point, it immediately is attracted by this local minimum, regardless of its global quality.

In the sequel, it is assumed that all integer variables are binary variables, *i.e.*, they are restricted to the values 0 or 1. This is not an essential restriction. Any integer variable with known upper- and lower bound can be represented by a number of binary variables, *e.g.*, the variable $z \in \{0,1,\cdots,15\}$ can be represented by the binary variables $x_1,\cdots,x_4$:

$$z = \sum_{i=1}^{4} x_i 2^{i-1}.$$

In most practical applications, the most efficient way of modeling is using binary variables. In our fuel management models for example, we might use variables

$z_{\ell,m}$ = the number of the node in which the bundle of age ℓ from trajectory m is placed,

but then the rest of the model is much more complicated as when using the binary variables

$$x_{i,\ell,m} = \begin{cases} 1 & \text{if bundle } (\ell,m) \text{ is placed in node } i; \\ 0 & \text{otherwise.} \end{cases}$$

In the sequel of this section we will discuss different techniques for integer optimization in more detail. We start with a discussion of *rounding heuristics.* Usually, these heuristics give only a local optimum, with no guarantee on global optimality of the solution. An other approach is the *Branch-and-Bound* method, that yields global optimality for convex problems. Branch-and-Bound is a widely used technique, especially for linear problems, with very efficient implementations. A third technique is *Outer Approximation.* When applied correctly, this technique gives in the limit a global optimum for convex optimization problems. It repeatedly uses the linear Branch-and-Bound method as a subproblem optimization technique. The fourth technique is the *addition of cuts*. This is widely used for linearly constrained problems as well. Some textbooks about (mixed) integer optimization are [90, 96, 114]. An annotated bibliography on the subject is also available [26]. Most textbooks concentrate on linear mixed-integer optimization, much fewer textbooks give a specific overview of nonlinear mixed-integer optimization algorithms [34].

4.3.1 Rounding heuristics

Rounding heuristics are seldom sufficient to find global optimal solutions, even for linear problems. Nevertheless, they do have practical value. In linear and convex integer problems, they may be used to find at least initial feasible solutions, whose objective value gives a bound on the truly optimal objective value: all solutions that have a worse objective value can be discarded. Since such bounds play a significant role to speed up other search processes like Branch-and-Bound and Outer Approximation, rounding heuristics are important tools in integer optimization.

There are different heuristic ways to obtain rounded integer solutions from fractional solutions. Starting from the variable with the largest non-integer value, one may fix variables to one, until some constraint is violated. As an alternative, one may start with the *smallest* nonzero value. In this way, an integer solution is obtained that is likely to be somewhat 'further away' from the fractional solution. In some cases, one might consider to use dual information. Commercial packages contain some rules of thumb as the two sketched above, to find good rounded solutions.

In some cases, it might be practically possible to evaluate all possible 0-1 combinations that can be constructed from a fractional solution. This is only true if

- the number of non integer variables is very small, and
- evaluation of the model with a fixed set of binary variables is not too costly.

In order to illustrate the first point, look at the classical assignment problem where we are looking for a binary matrix, such that each row sum and each column sum must be one. Suppose that a continuous optimization algorithm returns as a solution the following two-by-two submatrix:

$$\begin{pmatrix} .6 & .4 \\ .4 & .6 \end{pmatrix}.$$

There are only two possible rounded solutions in this case. In the case we have a full 3×3 fractional submatrix (9 non-integer variables), there are 6 possible rounded solutions, the 5×5 case with 25 non-integer variables already gives 120 rounded solutions, while two independent blocks of this size in the matrix give $120^2 = 14400$ solutions. Calculating the objective values for such a large number of solutions, that are very close to each other in the solution space, is not efficient, particularly not in the scope of reload pattern optimization, where evaluating one solution is costly.

Suppose that all rounded solutions are evaluated, and the one with the best objective value is selected. If this objective value is the same as the objective value of the fractional solution, then it is –at least in linear and convex problems– guaranteed to be globally optimal. In general however, the evaluation of all possible roundings of the fractional optimum does not guarantee global optimality, even not in the linear or convex case. For example, in the plain model of Figure 4.1, the global optimum is not a rounding of the continuous optimum.

4.3.2 The Branch-and-Bound method

A more elaborate strategy to derive integer solutions is the use of a Branch-and-Bound method. For convex problems, such a method leads to global optimal solutions. For nonconvex problems, this still may be true, but then special adjustments have to be made, as will be discussed later in this section. The explanation in this section assumes that the direction of the optimization is minimization.

The idea of Branch-and-Bound is to reduce the problem to a number of smaller subproblems. In each subproblem information is obtained about the best possible solution value in this subproblem (a *lower bound* on the solution value), and the current best solution value found in this subproblem (an *upper bound* on the solution value). If the lower bound in one subproblem is worse than the upper bound already obtained in another subproblem, the first subproblem never will contain the best possible solution, so it needs not to be explored further and can be fathomed.

In order to explain the Branch-and-Bound method in more detail, let us consider a convex mixed-integer optimization problem, where all integer variables are binary. The integer variables are denoted by $x_1, \cdots, x_n$. The Branch-and-Bound method maintains a *node table*. This is a list of problems where some of the binary variables are fixed to 0 or 1. The solution of the node is a lower bound on all integer solutions that can be obtained with the currently fixed subset of binary variables. Possibly there is also an upper bound that may be obtained with an integer rounding of the solution at this node. The best integer solution value that is found up to the current iteration in the process is denoted as Upb^*. In the description below, it is assumed for simplicity of exposition that all generated solutions are feasible. The algorithm proceeds as follows.

1. Start with an empty node table, let x^* be a vector that will contain the global best solution, and let $Upb^* = \infty$.

2. Solve the relaxed problem, in which all binary variables x_i are relaxed to allow values $0 \leq x_i \leq 1$.

3. If all binary variables happen to be 0 or 1, then the solution is the global optimal solution, update x^* and Upb^* and go to step 8.
   Otherwise add the current problem to the node table. Optionally: calculate a rounded solution. If it has a lower solution value than Upb^*, then store it as x^*, and update Upb^*.

4. *Start of loop.* If the node table is empty go to step 8.

5. *Node and variable selection.* Select a node and remove it from the node table. In this node, select a not-yet-fixed binary variable. Create two new nodes where the selected variable is fixed to 0 and 1, respectively. For each of these nodes perform the following step:

6. *Find new bounds.* Solve the relaxed problem to obtain a lower bound. If this lower bound is above the current Upb^* then delete this node.
   Otherwise, if the solution is integral then store it as x^* and update Upb^*. If the solution is not integral (but better than Upb^*) then store the node in the node table. Optionally, an upper bound on the solution in this node may be calculated by rounding. If this leads to a solution with a solution value lower than Upb^*, then store it as x^* and update Upb^*.

7. *Update node table.* Delete all nodes from the node table that have a lower bound that is larger than or equal to Upb^*. Go to step 4.

8. Return the solution corresponding to Upb^* as the best solution.

The procedure is illustrated by an artificial example in Figure 4.6. Starting from the relaxed optimum with value 50, a solution is found with upper bound 100. Variable x_2 is selected for branching, resulting in node 2 and 3. Node 2 is branched on x_1, and an integer solution with value 75 is found in node 6. Therefore nodes 4 and 5 are fathomed immediately, since their lower bound is worse than the best known upper bound. Node 8 can be fathomed as soon as the upper bound of node 10 is known.

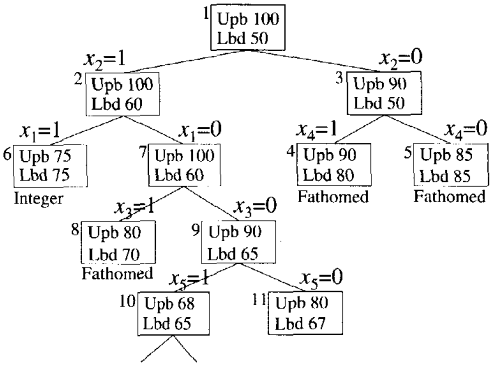


FIGURE 4.6: Example Branch-and-Bound tree.

The power of the algorithm lies in the ability to cut off branches as early as possible. It is therefore important to get good rules how to select the next node to branch on, and how to select the branching variable. Furthermore, it is important to obtain good (tight) lower and upper bounds. If bad choices are made, theoretically it can happen that all nodes in the tree have to be expanded, in which case the amount of work is even larger than for complete enumeration. Usually, however, the computational burden reduces drastically when compared to complete enumeration.

This is equal to a complete enumeration of all possible integer assignments, which is only computable for very small problem sizes.

For nonconvex problems, this algorithm in its standard form is not able to produce global optimal solutions. The algorithm is based on the assumption that the relaxed solution is a true lower bound of all possible solutions below the current node. For nonconvex problems we are in general not able to produce global minima in the nodes, hence this assumption does not hold. A node that is fathomed because of its bad lower bound, may still have another local optimum with a better lower bounding value, or it may even contain the global optimum.

There exist Branch-and-Bound-based methods that are designed to find global optimal solutions for nonconvex problems. These methods require not only a relaxation and fixing of the integer variables, but also convex approximations of the nonconvex feasible set and/or objective function, and a partitioning scheme to divide the feasible set into smaller parts in the subsequent nodes. These nonconvex Branch-and-Bound methods are discussed in Section 4.4.

4.3.3 Outer Approximation

Outer Approximation (OA) is another method for nonlinear mixed- integer optimization. It can be viewed as an extension of the Benders decomposition method for linear mixed-integer optimization. In the basic form as developed by Duran and Grossmann [30, 34], it solves problems of the following form:

$$\begin{aligned} \text{minimize} \quad & f(x)+c^T y \\ \text{subject to} \quad & g(x)+By \leq 0 \\ & A_1 x \leq a_1 \\ & A_2 y \leq a_2 \\ & y \text{ is a binary vector,} \end{aligned}$$

where $f(x)$ can be a nonlinear function, and $g(x)$ a vector of nonlinear functions. The most important restrictions are:

1. the binary variables are only involved in linear terms;
2. no nonlinear equality constraints are present;
3. the nonlinear functions f and g are convex.

All restrictions are violated by the fuel management optimization problem. The first restriction can be circumvented by a sort of implementation trick: a copy y' of the binary vector y is added to the problem. In all nonlinear terms where y occurs, it is replaced by y', and the linear constraints

$$y = y'$$

are added. A more elegant way to work around this restriction exists [33], but was not available in the program that we have used in our computational studies. The second restriction also is not essential when using Equality Relaxation (ER), as will be explained later in this section. Similarly, it will be illustrated that the third restriction is relaxed by the use of Augmented Penalty (AP), although no guarantee of global optimality of the solution will be possible in the case that the function f and/or some of the functions in g are not convex.

The basic idea of OA is that in each iteration, a lower and upper bound on the MINLP solution are generated. After solving a first NLP (where the integer restriction on y is relaxed), it is determined whether the solution is integer. If so, then the solution is optimal and the algorithm is terminated. Otherwise, a sequence of iterations is started, where two problems are solved in the kth iteration:

- an incomplete master problem, that provides a lower bound on the minimum possible solution value. This is a Mixed-Integer Linear Optimization problem (MILP), resulting in a solution (x, y^k).
- a primal problem, being the original problem where the y-variables are fixed to y^k. This problem is a continuous NLP in the x-variables, resulting in a solution x^k. If the solution (x^k, y^k) is feasible, it is an upper bound to the optimal solution.

The incomplete master problem is based on the idea that convex functions can be underestimated by linear functions, *cf.* Figure 4.7. The linear functions are obtained by first order Taylor approximations. The feasible area is completely described by the (infinite) set of linear approximations in each point, but of course it is impossible to include all these inequalities in a practical implementation. The incomplete master problem contains the linear approximations around the solution points of the previous NLP subproblems. In the first iteration, the problem is linearized around the solution point of the initial NLP, and the MILP master problem is solved using a MILP solver. This is usually a linear simplex or interior point solver, embedded in a linear Branch-and-Bound framework. This results in a solution where all y-variables are integer. However, since it is a solution to the linearized problem, the x-variables need not satisfy

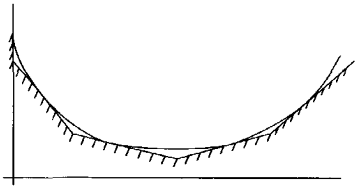


FIGURE 4.7: Convex region estimated by linear inequalities.

the nonlinear inequality constraints, or they may not be optimal with respect to the objective function. Therefore the primal problem has to be solved.

If the solutions of the master problem and the primal problem differ by less than the required accuracy, the solution for convex problems is optimal. Another heuristic stopping criterion is to stop whenever the solution of a NLP primal problem becomes worse than the previous NLP solution. This heuristic may in practice stop too early.

As is mentioned earlier, the method needs to be adapted to handle equality constraints and nonconvex problems. Equality Relaxation (ER) uses Lagrange multipliers from the NLP primal solution in order to derive a correct linearization. Suppose the NLP contains a nonlinear constraint

$$h(x) = 0. \tag{4.15}$$

If the Lagrange multiplier in the optimal solution is zero, then this constraint was apparently not active, and no linearization of this constraint is to be added to the master problem. Otherwise let σ be the sign of the Lagrange multiplier, then the current solution point would remain a local optimal solution if constraint (4.15) was replaced by the constraint

$$\sigma h(x) \leq 0.$$

This constraint may be linearized in the same way as other inequality constraints (assuming that it satisfies convexity properties).

If nonconvex inequality constraints are present, or result from the ER step, or when the objective function is nonconvex, the linearization step may wrongly cut off parts of the feasible area, and may cut off the global optimal solution. With the Augmented Penalty (AP) technique, the linear constraints are relaxed by some penalty variable. For example, around the kth NLP solution, the linearized inequality constraint

$$g(x^k) + \nabla g(x^k)(x - x^k) \leq 0$$

is replaced by

$$g(x^k) + \nabla g(x^k)(x - x^k) \leq p_k,$$

where p_k is a penalty variable that is added to the objective function with some positive penalty parameter. In this (heuristic) way, solutions that do not satisfy the linearized constraints can be reached if their objective function value is very good.

The case when binary variables occur in nonlinear terms can be treated by the Generalized Outer Approximation scheme (GOA), proposed in [33]. In this case, the primal problem is not only linearized with respect to the continuous variables, but also with respect to the binary variables. This means that the incomplete master problem may return solutions that are infeasible with respect to the y-variables. The GOA introduces extra linear cuts to handle this case.

4.3.4 Cutting Planes

This technique is usually applied to MILP problems. However, it may be helpful to speed up the solution process of MINLP problems.

To understand the idea of cuts, we recall Figure 4.1. The reason why a non-integer solution is found by a continuous linear optimization algorithm is that the feasible area of the continuous relaxation is not equal to the *convex hull* of the integer feasible points. This convex hull is defined as the smallest polytope that includes all integer feasible points. In the cutting plane technique, one adds new constraints, so called 'cuts' to the relaxed problem, so that they give a tighter approximation of the convex hull. Consider for example the problem

$$\begin{aligned} \text{minimize} \quad & -5x_1 + x_2 \\ \text{subject to} \quad & x_1 + x_2 \geq 6 \\ & 9x_1 + 5x_2 \leq 45 \\ & x_1, x_2 \geq 0, \text{ integer.} \end{aligned} \tag{4.16}$$

The feasible area of this problem is given in Figure 4.8. The relaxed problem as such would

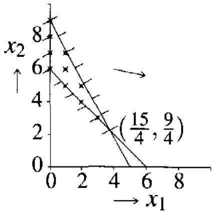


FIGURE 4.8: Example where cuts can be derived.

return the optimal solution $(\frac{15}{4}, \frac{9}{4})$. Multiplying the first constraint of (4.16) by 5 however, gives that

$$5x_1 + 5x_2 \geq 30.$$

Combining this with the second constraint immediately yields

$$4x_1 \leq 15 \Rightarrow x_1 \leq \frac{15}{4} \Rightarrow x_1 \leq 3,$$

since x_1 must be integer. Adding this constraint to the problem leads in our case to the optimal solution $(3,3)$.

As just illustrated, cuts can be derived by using information from the fractional solution. Also, a number of standard cuts exist for different classes of combinatorial optimization problems. Apart from these standard classes, implementations of cuts should be based on some problem knowledge in order to make them efficient. Otherwise, the situation may occur that subsequent cuts only cut off very small parts of the non-interesting region, converging only very slowly or not at all to the convex hull of the integer points.

In the same way as above illustrated for linear problems, cuts can be applied to nonlinear convex problems. For nonconvex problems, the addition of cuts can be dangerous. As illustrated by Figure 4.9, the global optimum can easily be cut off incorrectly. In a nonconvex problem, cuts should therefore be applied either on a local scale, for example only during one iteration, or based on insight into the problem. The latter type of cuts is frequently used in reload pattern optimization in the form of engineering constraints.

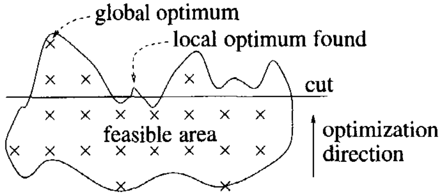


FIGURE 4.9: All better integer solutions are wrongly cut off.

In nonconvex problems such as the reload pattern optimization problem, cuts may also be used to escape from local optimal solutions. After a local solution is found, a cut may be applied that cuts off this local optimal solution, in the hope that re-optimization will lead to a better local optimum. As with integer cuts, such cuts should only be applied locally, or based on insight into the problem. In Chapter 5 we discuss such cuts that we have applied to improve results in reload pattern optimization.

4.3.5 Other techniques

Other proposed techniques to treat MINLP problems are *Generalized Benders Decomposition* (GBD) and *Generalized Cross Decomposition* (GCD). As OA, GBD is based on splitting the problem into a master problem and a primal (sub)problem, but it uses Lagrange multipliers for all constraints in setting up the incomplete master problem. GCD [53] splits the problem into a master problem, a primal and a dual subproblem. Since these two methods are not applied in our work, they are not treated here in more detail; we refer to [34] for an overview.

Combination of algorithms

All MINLP techniques described until now can be combined with neighborhood search heuristics. When the nonconvex mixed-integer optimization algorithm has stopped, a neighborhood search may be initiated from the current optimal solution. How and when this is an attractive option, is very much dependent on problem structure. In some of our reload pattern optimization models, several good integer solutions are situated close to each other. However, since they are all separate local optima in the continuous space, the MINLP solver stopped just in one of these pretty good solutions. After that, only a few iterations of a local search heuristic could find an improved solution. In this way, total computation time was reduced compared to a pure local search, while the solution quality was increased compared to both pure local search and pure MINLP.

4.4 Global optimization approaches

The techniques described so far aim at finding global optimal solutions for convex problems, For nonconvex problems, these techniques give no guarantee about the quality of the obtained local optimum. Global optimization algorithms are needed to give such a guarantee. In their most general form, they do require mild assumptions with respect to continuity or differentiability of the constraints and/or the objective function. They may use function and constraint evaluations as black box, or they may include some problem-specific elements such as relaxations, dual problems, *etc.* Using suitable branching and cutting strategies, either global optimal solutions are obtained, or the quality of solutions can be specified (*e.g.*, it is within ...% from the global optimum).

As with Branch-and-Bound, these methods create a number of smaller problems. However, there are some additional features needed to make Branch-and-Bound suitable for global optimization problems. In Branch-and-Bound for linear or convex mixed-integer problems, the *branching* step consisted of fixing an integer variable to different integer values in the different 'child' problems. In continuous global optimization, the subproblems are created by dividing the feasible area into different subspaces. The most straightforward way to do so is by dividing the range of a variable into smaller parts. The *bounding* is also different as with standard Branch-and-Bound. For linear or convex mixed-integer problems, a lower bound (in the minimization case) was found by solving a relaxed problem, where the integrality restrictions were relaxed. Rounding heuristics could be applied in a trial to derive an upper bound in each node. For nonconvex problems, simply solving relaxed solutions will not give true lower bounds, but only local optimal solutions. These solutions may be used to derive upper bounds, but in order to derive the lower bounds, other techniques are needed.

In this section, two of these techniques are discussed: partitioning based on the Lipschitz constant *(Lipschitz optimization)*, and *convex over-relaxation* combined with *range reduction*. It should be noted that these techniques only work for small problems. Depending on the problem structure and the required accuracy, the problem size that can be handled typically ranges from ten or twenty variables to several hundreds of variables. An overview of global optimization techniques is found in [55]. Lipschitz optimization is treated in depth in [102]. Branch and Reduce is discussed in [111].

4.4.1 Lipschitz optimization

This method is based on the concept of Lipschitz continuity. A function f is said to be *Lipschitz continuous* if there exists a constant L, called the *Lipschitz constant*, such that

$$|f(x_2) - f(x_1)| \leq L||x_2 - x_1||$$

for all two points x_1 and x_2 in the domain of f. An optimization problem is Lipschitz continuous if both the objective function and all constraint functions are Lipschitz continuous. For differentiable functions, this means that there is an upper bound L on the absolute value of the derivatives, which can be used to derive bounds on function values.

For illustration, consider the one-dimensional nonconvex objective function of Figure 4.10a that has to be minimized over the interval $[a,b]$. Suppose that at some stage during the search the objective function is evaluated in the points 1 through 9. Then it is known that the objective function is underestimated by the sawtooth curve as indicated. The problem can therefore be reduced to 8 smaller subminimization problems over the intervals $1-2, 2-3, \cdots, 8-9$, and in each interval a lower bound on the function is obtained by the minimum of the sawtooth. Since the computed objective function in point 5 is better than the sawtooth-minimum in the intervals 2-3, 3-4 and 6-7, these intervals can be discarded from the search. Lipschitz optimization algorithms contain rules how to select the next point to be evaluated, so that the partitioning is as efficient as possible, and large parts of the feasible solution area can be discarded as early as possible.

The advantage of Lipschitz optimization methods is that exact derivative information is not needed, except for the Lipschitz constant. On the other hand, much partitioning may be needed before a reasonable approximation is obtained, especially in multi dimensional cases. A more efficient implementation is possible if more information on the functions to be evaluated is available. It may for example be known that in parts of the solution space the maximum derivative value is much smaller, so that the Lipschitz constant then may locally be set to a smaller value. Without such adaptations, Lipschitz optimization is only possible for at most a few tens of variables.

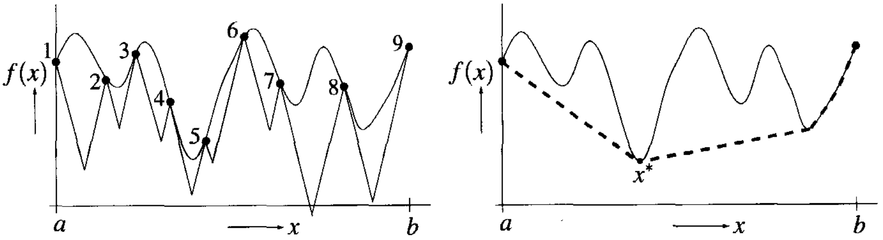


FIGURE 4.10: Underestimation of a nonconvex objective function: a) using the Lipschitz constant; b) using the convex envelope.

4.4.2 Convex relaxations within Branch and Reduce

As soon as some structural information on the objective and constraint functions is known, one may derive better upper- and lower bounding functions as can be obtained by Lipschitz optimization. In the example of Figure 4.10, the objective function can be underestimated by the dashed function in Figure 4.10b. If an analytical description of this tight underestimating function were known, the global minimum x^* could be found by just minimizing this function using standard NLP techniques. In general, such an *exact* convex underestimating function is not known, but a much weaker convex underestimation can be derived. The idea of Branch and Reduce is that we can find tighter convex underestimations if the considered variable intervals become smaller. So if we found a minimum of an underestimating function, we may divide the range of some variable into the two parts left and right of this minimum, and derive tighter underestimators in these two subproblems.

A tight convex lower bounding function as in Figure 4.10b is known as the *convex envelope* of the function. A very general and complex scheme for deriving convex envelopes for analytical functions, products of functions and functions of functions is derived in [81]. The practical value of this scheme is limited to very special structures. Application of the scheme to bilinear two-dimensional functions gives a convex underestimating function composed of two linear parts, as is shown in Figure 4.11. Here the bilinear function is

$$z = f(x,y) = x * y$$

and the tightest convex underestimation of this function on the unit square is given by the two inequalities

$$\begin{aligned} z &\geq 0, \\ z &\geq x + y - 1. \end{aligned}$$

Trilinear functions already lead to tight convex underestimations that contain at least 8 of such linear inequalities.

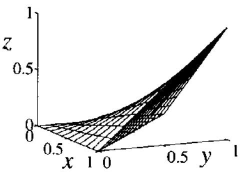


FIGURE 4.11: Bilinear function can be underestimated by 2 linear functions.

Another way to derive a convex underestimating function is by adding a convex quadratic function to the nonlinear function, such that the resulting function also becomes convex within the specified region, assuming that the function is two times continuously differentiable [4]. This is done by considering the Hessian of the function. Addition of quadratic terms adds values to the diagonal elements of the Hessian. Quadratic functions are added with a multiplier such that the resulting combined Hessian becomes positive definite within the required region.

Thus far, we only considered a nonlinear objective function. In case we have to relax a nonlinear constraint it depends on the type of constraint what type of estimating functions are needed. A 'greater than' constraint needs a convex underestimating function. A 'lower than' constraint needs a concave overestimating function, while an equality constraint has to be estimated by a 'box' defined by both types of estimations. Suppose for example that the bilinear function in Figure 4.11 represents an equality constraint

$$z = xy.$$

This constraint is then relaxed by four linear inequalities, that together enclose the nonlinear shape in a 'box':

$$\begin{aligned} z &\geq 0, \\ z &\geq x+y-1, \\ z &\leq x, \\ z &\leq y. \end{aligned}$$

This box is tighter when the range over which the variables are defined is smaller. So, observe that:

1. using convex underestimates and concave overestimates leads to a convex optimization problem, the minimum value of which provides a lower bound on the global optimum of the nonconvex problem (note that we consider a *min*imization problem);
2. this lower bound can be tightened if the range of one or more variables is divided into two or more parts, and for each part separately the relaxed problem is solved;
3. when in one of these parts a solution of the 'real' problem is known with a better objective value than the lower bound in another part, this other part cannot contain the global optimum and can be fathomed.

The Branch and Reduce algorithm is essentially a Branch-and-Bound algorithm, with some dedicated Branching rules. The extra feature of Reduction consists of a set of rules to further reduce the variable ranges in the nodes. This can again be illustrated with Figure 4.11. Suppose that in some node, the ranges of x and y are both reduced to $[0, 0.5]$. Then it can be derived that the range of z is immediately reduced to $[0, 0.25]$. This in turn may possibly be used to tighten some relaxation of another constraint.

Using Branch and Reduce, one may identify subregions that cannot contain the global optimum. On the other hand, it also gives quality bounds on solutions that are obtained. During the search, the global lower bound increases, while the global upper bound (*i.e.*, the current best found solution) decreases. The gap between the two is an indication how far we are from the global optimum. For small and nicely structured problems, this gap can be driven towards zero (within some accuracy), resulting in a guaranteed global optimal solution.

4.5 Summary

In this chapter mixed-integer nonlinear programming techniques were discussed that may be useful to solve the fuel management optimization problem. Two main streams are identified.

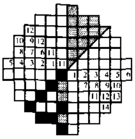
One stream combines gradient-based nonlinear optimization algorithms with special techniques to find integer solutions. This approach is appropriate to find good quality local optimal solutions in a reasonable amount of time.

The second stream consists of global optimization algorithms. These algorithms result in a guaranteed global optimal solution, but are only applicable for very small (tens up to at most a few hundred variables and constraints), or very specially-structured problems.

For both approaches, the formulation of the model is very important. Different representations of basically the same model may result in much easier or harder nonconvexity structures, or in harder or in less severe numerical difficulties. Also, the specific algorithm of choice depends very much on problem structure. Regarding the solver for the nonlinear optimization part, for example, the only way to decide between the different algorithms is often by experimentation.

It must be remarked that the way in which the optimization algorithm is implemented has significant influences on the performance of the algorithms. An efficient and robust implementation of an optimization algorithm must satisfy some basic properties. It should make use of sparse linear algebra. This is especially important for large problems that have lots of structural zeros in the constraint gradients, a property that holds for the reload pattern optimization problem. Variable bounds are very special constraints that can be handled efficiently in a number of algorithms. The implementation should offer the possibility to specify these bounds separately. Efficient implementations of nonlinear optimization algorithms also treat linear constraints separately, so that expensive and less robust techniques for nonlinear optimization need not be applied for these constraints. Implementations of integer optimization algorithms may treat binary variables apart from general integer variables, since the first type can be handled more efficiently. The handling of numerical errors is another important implementation issue for all algorithms. There are still lots of other properties that will influence the quality of an algorithm in a specific implementation.

For all algorithms, it is advantageous to exploit specific properties of the problem at hand. In the next chapter, we will explain our models in detail, what relaxations and cuts we implemented, what starting points we used, and other problem-specific implementation issues that were needed to obtain our results.



Chapter 5

Implementation issues in mixed-integer nonlinear optimization

This chapter gives a detailed description of our implementations to solve the fuel management optimization problem. We use the models developed in Chapter 2, which we are going to solve with the optimization methods as described in Chapter 4. Lots of problem-specific issues are involved in the actual implementation. In contrast to linear or convex optimization models, good results cannot be expected when the models are implemented in a standard available mixed-integer nonlinear optimization package in a straightforward way. If done so in one of the few commercially available packages for MINLP, the solver stops in nearly all test problems without finding any solution. Even the standard continuous relaxation cannot be solved with standard NLP packages, without the additional effort as described in this chapter.

In order to get MINLP algorithms to work for the fuel management problem, one has to take care about a lot of things. First of all, the continuous NLP relaxation should give good quality local optimal solutions. Some issues that we looked at are:

- The starting point.
- Variable bounds.
- Addition of extra (linear) constraints.
- Relaxation of constraints and addition of objective penalties.
- Choice of discretization patterns.
- Reformulations to reduce nonconvexity.
- Engineering constraints.

The computation of a starting point is discussed in Section 5.1. The other items regarding the nonlinear part are discussed in Section 5.2. After obtaining a local optimal continuous solution, an integer solution must be obtained. In Section 5.3 we deal with the integer part. We consider specific properties of implementations using rounding heuristics, Outer Approximation and Branch-and-Bound. Also, our implementation of a pairwise interchange algorithm is described, and the combination of it with MINLP algorithms. Another important part of our research was the implementation of cuts that are used to find improved local optimal solutions. Section 5.4 describes the various types of cuts that we implemented. Finally, Section 5.5 describes in more detail the solvers used and the test environments.

Before continuing, we repeat a complete listing of the basic model as given in Section 2.3.1, since we will refer repeatedly to equations in this model.

$$\underset{x,k^\infty,R,k^{\text{eff}}}{\text{maximize}}\ k_T^{\text{eff}} \tag{5.1}$$

subject to

the linear constraints:

$$\sum_{i=1}^{I} V_i x_{i,\ell,m} = 1, \qquad \ell=1,\cdots,L,\ m=1,\cdots,M \tag{5.2}$$

$$\sum_{\ell=1}^{L}\sum_{m=1}^{M} x_{i,\ell,m} = 1, \qquad i=1,\cdots,I \tag{5.3}$$

the nonlinear constraints:

$$\begin{aligned} k_{i,1}^\infty &= \sum_{m=1}^{M} x_{i,1,m} k^{\text{fresh}} \\ &+ \sum_{\ell=2}^{L}\sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} V_j x_{j,\ell-1,m} k_{j,T}^\infty, \qquad i=1,\cdots,I \end{aligned} \tag{5.4}$$

$$k_t^{\text{eff}} R_{i,t} = \sum_{j=1}^{I} G_{i,j} k_{j,t}^\infty R_{j,t}, \qquad i=1,\cdots,I,\ t=1,\cdots,T \tag{5.5}$$

$$\sum_{i=1}^{I} V_i k_{i,t}^\infty R_{i,t} = 1, \qquad t=1,\cdots,T \tag{5.6}$$

$$k_{i,t+1}^\infty = k_{i,t}^\infty - \alpha\Delta_t k_{i,t}^\infty R_{i,t}, \qquad i=1,\cdots,I,\ t=1,\cdots,T-1 \tag{5.7}$$

$$k_{i,t}^\infty R_{i,t} \le \frac{f^{\text{lim}}}{\sum_{j=1}^{I} V_j}, \qquad i=1,\cdots,I,\ t=1,\cdots,T \tag{5.8}$$

the variable bounds:

$$k_t^{\text{eff}} \ge 0, \qquad t=1,\cdots,T \tag{5.9}$$

$$k_{i,t}^\infty, R_{i,t} \ge 0, \qquad i=1\ldots I,\ t=1\cdots T \tag{5.10}$$

$$x_{i,\ell,m} = x_{j,\ell,m}, \qquad \text{all adjacent diagonal pairs } (i,j),\ \ell=1,\cdots,L,\ m=1,\cdots,M \tag{5.11}$$

$$x_{i,\ell,m} \in \{0,1\}, \qquad i=1,\cdots,I,\ \ell=1,\ldots,L,\ m=1,\cdots,M. \qquad (5.12)$$

It turns out to be convenient to introduce some vector notation. We define

$$k^{\infty}_{\cdot,t} = (k^{\infty}_{1,t},\cdots,k^{\infty}_{I,t})^T,\ \ t=1,\cdots,T$$
$$R_{\cdot,t} = (R_{1,t},\cdots,R_{I,t})^T,\ \ t=1,\cdots,T.$$

5.1 Starting point and fixed core evaluation

The choice of the starting values can have a big influence on the behavior of the algorithms. This influence roughly breaks up into two parts.

1. The initial values of the assignment variables $x_{i,\ell,m}$ have influence on the final local optimal solution. Using some engineering knowledge of the problem, a good starting point should be selected from which with high probability a good local optimal solution can be obtained.
2. The model contains many nonlinear and nonconvex equalities. Even if correct values for the assignment variables are supplied, it may be a difficult task for an optimization routine to find feasible values for the physical variables from the model equations. It is therefore important to supply values for the physical variables such that at least the nonlinear equalities are satisfied as close as possible.

5.1.1 *Initial values of the assignment variables*

Due to the nonconvexity of the problem, it is desirable to use a starting value that is in the 'vicinity' of good solutions. A good initial solution could help to avoid that the optimization will trap into a poor local optimal solution. To this end, engineering knowledge is used to determine some basic properties of good loading patterns. It is known that fresh bundles in the middle lead to a (too) high power peak. On the other hand, the bundles on the boundaries of the core should be sufficiently burned, since otherwise too many neutrons are lost in the surrounding water. It is known that good loading patterns therefore have a sort of 'ring structure', in which old bundles are put on the boundaries, fresh bundles somewhere between the core center and the periphery, and huge isles of fresh bundles should be avoided since they cause a too high power peak. An example of a good loading pattern satisfying these properties is given in Figure 5.1.

In the models that are used so far, it is not a good idea to use an integer solution as a starting point, since many integer solutions are local optimal solutions in the space of continuous variables. An NLP algorithm never escapes from such a local maximum. As an alternative, fractional starting patterns are used. For each age group, a number of likely positions for the bundles of this age are located. This number may be larger than the number of bundles itself, as is the case in Figure 5.2. Using these locations, a fractional loading pattern is constructed. This fractional assignment is made in such a way that a fraction ω of the volume of the bundles of age ℓ is spread in the nodes assigned for this age, and $1-\omega$ is spread over the remaining nodes.

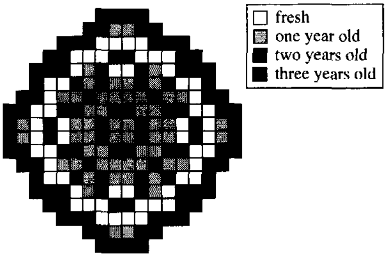
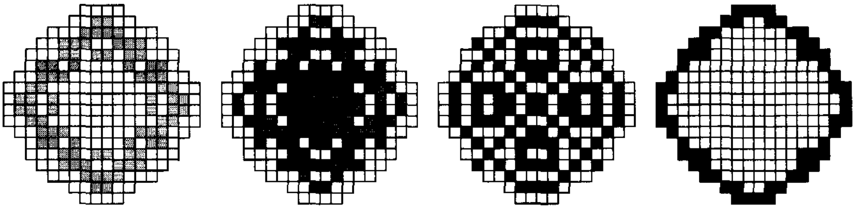


FIGURE 5.1: Good loading pattern with ring structure.



Age group 1 Age group 2 Age group 3 Age group 4

FIGURE 5.2: Most likely locations for bundles in the different age groups.

Since the number of likely locations for an age may be larger than the number that is actually needed, a whole range of possible fractional starting patterns satisfying these rules might exist. We developed two different models to find a balanced distribution of the bundles over the nodes. Both models are convex quadratic optimization models on their own. The models do not make distinction between the different bundles of the same age, they use only the variables

$$\hat{x}_{i,\ell} = \text{the total volume of fuel of age } \ell \text{ to be located in node } i.$$

After finding values for these variables, the actual assignment variables can be found using

$$x_{i,\ell,m} = \frac{\hat{x}_{i,\ell}}{M}.$$

The candidate positions for the different ages are given by the parameter $c_{i,\ell}$:

$$c_{i,\ell} = \begin{cases} 1 \text{ if } i \text{ is a likely location for age } \ell, \\ 0 \text{ otherwise.} \end{cases}$$

Using ω, $c_{i,\ell}$ and $\hat{x}_{i,\ell}$ the following models can be defined.

Starting point model I

In this first model, we require that $c_{i,\ell}$ is defined in such a way that exactly a fraction ω of the bundles of each age ℓ can be assigned to the positions i: $c_{i,\ell} = 1$. The fuel is evenly distributed by minimizing the sum of squared assignment values.[1]

$$\begin{array}{llll} \underset{\hat{x}}{\text{minimize}} & \sum_{i=1}^{I}\sum_{\ell=1}^{L} \hat{x}_{i,\ell}^2 & & \\ \text{subject to} & \sum_{i=1}^{I} c_{i,\ell} V_i \hat{x}_{i,\ell} & = \omega M, & \ell = 1,\cdots,L \\ & \sum_{i=1}^{I} V_i \hat{x}_{i,\ell} & = M, & \ell = 1,\cdots,L \\ & \sum_{\ell=1}^{L} c_{i,\ell} \hat{x}_{i,\ell} & = \omega, & i = 1,\cdots,I \\ & \sum_{\ell=1}^{L} \hat{x}_{i,\ell} & = 1, & i = 1,\cdots,I \\ & 0 \le \hat{x}_{i,\ell} & \le 1, & \mathrm{i} = 1,\cdots,I,\ \ell = 1,\cdots,L. \end{array} \tag{5.13}$$

This model gives quite evenly distributed patterns, but it might be infeasible for some candidate position set $c_{i,\ell}$. The second model is devised to overcome this problem.

Starting point model II

$$\begin{array}{llll} \underset{\hat{x}}{\text{minimize}} & \sum_{i=1}^{I}\sum_{\ell=1}^{L} \omega(\hat{x}_{i,\ell} - c_{i,\ell})^2 + (1-\omega)(\hat{x}_{i,\ell} - (1 - c_{i,\ell}))^2 & & \\ \text{subject to} & \sum_{i=1}^{I} V_i \hat{x}_{i,\ell} & = M, & \ell = 1,\cdots,L \\ & \sum_{\ell=1}^{L} \hat{x}_{i,\ell} & = 1, & i = 1,\cdots,I \\ & 0 \le \hat{x}_{i,\ell} & \le 1, & \mathrm{i} = 1,\cdots,I,\ \ell = 1,\cdots,L. \end{array} \tag{5.14}$$

This model always returns a feasible solution, even if an exact match of the fraction ω to the candidate positions is not possible. On the other hand, for cases where an exact match is possible, the solution may somewhat deviate from this exact match. This is not a problem, since both ω and the ring structure as defined by $c_{i,\ell}$ provide only an approximation of a good starting point. The most important issue is to find a reasonable good starting point in a clever way, but there is no rule how to select the best one.

---

[1] Instead of the listed objective value, one also might minimize the sum of $(\hat{x}_{i,\ell} - c_{i,\ell})^2$. Although the resulting pattern may be different, it also gives an evenly spread distribution.

5.1.2 Initial values of the physical variables

In order to make the nonlinear optimization robust, it is not enough that the assignment variables have some sensible initial value. If all physical variables in our model are set to 0 or to another bad value, nonlinear optimization solvers usually get stuck in a local infeasible solution[2]. Better starting values have to be supplied, that have some sensible physical meaning. Given a set of x-variables that satisfies (5.2)–(5.3) there exists exactly one nonnegative solution of (5.4)–(5.7). This nonnegative solution is computed by the method for fixed core evaluation as outlined in Section 5.1.3. In this way, we obtain a solution that is feasible with respect to all constraints of our optimization model except possibly the power peaking constraints (5.8) and the integrality restrictions (5.12). In general, the violation of the power peaking constraints is not a big problem, especially not when they are relaxed as is described in Section 5.2.3.

5.1.3 Fixed core evaluation

In the starting point calculation, and in several other algorithms such as the rounding algorithm or the pairwise interchange algorithm, we have to compute the values of the physical variables for a given set of assignment variables $x_{i,\ell,m}$ and, in case of the burnable poison (BP) model, for a given set of $\Sigma_{\mathrm{a},m}^{\mathrm{p,fresh}}$ values. This is done by solving the system of equations (5.4)–(5.7) in the basic model iteratively, and the system of equations (2.49)–(2.55) in case of the BP model. Due to the nature of the equilibrium cycle, the values $k_{i,1}^{\infty}$ are initially known for the fresh bundles. In order to find the other ones, we completely fill the core with fresh bundles and calculate the behavior of the core during the cycle. At the end of the cycle, we apply a reloading and calculate the next cycle. In this way, convergence to the equilibrium cycle is achieved. Usually, this convergence is obtained in about 10 cycles; in order to be on a safe side, we apply 19 reloadings, thus computing 20 cycles. In each cycle, we have to calculate the values of the physical variables in a number of time steps in the following order:

1. Given $k_{i,t}^{\infty}$ and $\Sigma_{\mathrm{a},i,t}^{\mathrm{p}}$ for all i, we compute $R_{i,t}$ and k_t^{eff} using the kernel equation (5.5). Since this equation reduces to a standard eigenvalue equation for fixed $k_{i,t}^{\infty}$, this is done by using the power method. Since we know that the largest eigenvalue in which we are interested has a strictly positive eigenvector, any strictly positive initial value for R converges to the eigenvector of interest. This eigenvector is normalized using (5.6).

2. Given $R_{i,t}$ we can compute $k_{i,t+1}^{\infty}$ and $\Sigma_{\mathrm{a},i,t+1}^{\mathrm{p}}$ for all i using the burnup equations (5.7) in the basic model, and equations (2.54), (2.55) in the BP model. In reactor physical terms, this is the depletion step.

3. At end of cycle, a reloading is performed using (5.4) in the basic model, and (2.49), (2.50), (2.51) in the BP model.

The procedure for the BP model is given in Algorithm 5.1. For reasons of programming logic, the reloading is performed at the beginning of each cycle instead of after each cycle. As the

---

[2] The value 0 is particularly bad in our case, since the first derivatives in all bi- and trilinear terms become zero. In such a situation it is very difficult to obtain improving directions.

```
Procedure EvalFixed($x, f_x$) {Burnable poison model}

if $k^\infty$ undefined then {No previous solution in memory}
    $\forall i:\ k^\infty_{i,EoC} := k^{\text{fresh}};\ R_{i,1} := 1.0;$
endif;
$\varepsilon^{\text{pow}} := 1.0e-5;$
for $c := 1$ to 20 do
    {Perform reloading}
    $\forall i:\ \overline{k}^\infty_{i,1} := \sum_{\ell=2}^{L} \sum_{m=1}^{M} \sum_{j=1}^{I} (x_{i,\ell,m} x_{j,\ell-1,m} \overline{k}^\infty_{j,T}) + (\sum_{m=1}^{M} x_{i,1,m}) k^{\text{fresh}};$
    $\forall i:\ \Sigma^{\text{p}}_{\text{a},i,1} := \sum_{m=1}^{M} x_{i,1,m} \Sigma^{\text{p,fresh}}_{\text{a},m};$
    $\forall i:\ k^\infty_{i,1} := \frac{\overline{k}^\infty_{i,t}}{1 + \Sigma^{\text{p}}_{\text{a},i,t}};$
    {Start core analysis}
    for $t := 1$ to $T$ do
        $k^{\text{eff}}_t := \sum_{i=1}^{I} \sum_{j=1}^{I} \frac{V_i G_{i,j} k^\infty_{j,t} R_{j,t}}{\sum_{i=1}^{I} V_i R_{i,t}};$
        repeat {Start power iterations}
            $k^{\text{eff}}_{\text{old}} := k^{\text{eff}}_t;\ \forall i:\ R^{\text{old}}_i := R_{i,t};$
            {Compute eigenvector}
            $\forall i:\ R_{i,t} := \sum_{j=1}^{I} G_{i,j} R_{j,t} k^\infty_{j,t};$
            {Compute eigenvalue}
            $k^{\text{eff}}_t := \frac{\sum_{i=1}^{I} \sum_{j=1}^{I} V_i G_{i,j} k^\infty_{j,t} R_{j,t}}{\sum_{i=1}^{I} V_i R_{i,t}};$
            {Normalize eigenvector}
            $\forall i:\ R_{i,t} := \frac{R_{i,t}}{\sum_{j=1}^{I} V_j k^\infty_{j,t} R_{j,t}};$
            $\Delta_1 := |k^{\text{eff}}_t - k^{\text{eff}}_{\text{old}}|;$
            $\Delta_2 := \max_i \left\{ \frac{|R_{i,t} - R^{\text{old}}_i|}{R^{\text{old}}_i} \right\};$
        until $\max(\Delta_1, \Delta_2) \le \varepsilon^{\text{pow}}/c;$
        if $t < T$ then
            Calculate_burnup
        endif
    endfor
endfor
$f_x := k^{\text{eff}}_T;$
```

ALGORITHM 5.1: Core evaluation routine for fixed loading pattern x.

cycles converge to the equilibrium cycle, the power method requires less and less iterations in the subsequent cycles, since the algorithm uses the $R_{i,t}$ values of the previous cycle as starting vector. When multiple fixed pattern evaluations are to be performed for patterns that are only slightly different, as is for example the case in PI, we re-use the $k^{\infty}_{i,t}$ and $R_{i,t}$ of the previous pattern as starting values for the current pattern. In this way, the algorithm converges faster to an equilibrium cycle, and the power method needs less iterations. Another implementation detail is that the convergence check of the power method depends on the cycle c. In this way, the initial cycles require less power iterations, while the final result still has the required accuracy.

The burnup calculation deserves special attention. When forward discretization is used, it is a straightforward implementation of (5.7), resp. (2.54), (2.55). The algorithm for central discretization is given in Algorithm 5.2. In this case we need to evaluate expressions of the form (5.29), that need values of $R_{i,t+1}$ in advance. Since the flux and the burnup rate are changing considerably during the first few cycles, the value $R_{i,t+1}$ of the previous cycle makes no sense. Therefore we apply forward discretization during the first few cycles. Only after four cycles, when all bundles that were initially in the core are replaced by new bundles, the solution becomes more stable and we switch to central discretizations.

```
Procedure Calculate_burnup {Central discretization}

if $c \leq 4$ then   {Forward differences}
    $\forall i:\ \overline{k}^{\infty}_{i,t+1} := \overline{k}^{\infty}_{i,t} - \alpha P_c \Delta_t \overline{k}^{\infty}_{i,t} R_{i,t}$;
    $\forall i:\ \Sigma^{\mathrm{p}}_{\mathrm{a},i,t+1} := \Sigma^{\mathrm{p}}_{\mathrm{a},i,t} - \alpha^{\mathrm{p}} P_c \Delta_t \Sigma^{\mathrm{p}}_{\mathrm{a},i,t} R_{i,t}$;
else   {Central differences (use prediction from previous cycle)}
    $\forall i:\ \overline{k}^{\infty}_{i,t+1} := \overline{k}^{\infty}_{i,t} \frac{1 - \frac{1}{2}\alpha P_c \Delta_t R_{i,t}}{1 + \frac{1}{2}\alpha P_c \Delta_t R_{i,t+1}}$;
    $\forall i:\ \Sigma^{\mathrm{p}}_{\mathrm{a},i,t+1} := \Sigma^{\mathrm{p}}_{\mathrm{a},i,t} \frac{1 - \frac{1}{2}\alpha^{\mathrm{p}} P_c \Delta_t R_{i,t}}{1 + \frac{1}{2}\alpha^{\mathrm{p}} P_c \Delta_t R_{i,t+1}}$;
endif
$\forall i:\ k^{\infty}_{i,t+1} := \frac{\overline{k}^{\infty}_{i,t+1}}{1 + \Sigma^{\mathrm{p}}_{\mathrm{a},i,t+1}}$;
```

ALGORITHM 5.2: Burnup calculation using central discretization.

5.2 The nonlinear part

In this section we discuss some model reformulations that help in solving the nonlinear part of the problem. Unless stated explicitly, these adaptations are based on the basic model without burnable poison. Not all of them can be used directly in the BP model. In the basic model, we can make use of the fact that the reactivity of each bundle, hence the effective multiplication factor, decreases monotonically with time. This property does not hold for the BP model, since initially the increase of reactivity due to BP burnup supersedes the decrease of reactivity due to fuel burnup.

5.2.1 Variable bounds

Variable bounds are tightened as much as possible. This helps the nonlinear optimization in staying feasible, since variable bounds give upper bounds on the step lengths in the algorithms. In this section, only bounds on the physical variables are discussed, since the assignment variables have the obvious bounds $0 \leq x_{i,\ell,m} \leq 1$. Upper and lower bounds on variables will be denoted by the superscripts $^{\mathrm{up}}$ and $^{\mathrm{lo}}$.

Bounding k^∞ and R

Clearly, an upper bound on k^∞ is given by k^{fresh}, the infinite multiplication factor of a fresh bundle, since k^∞ cannot increase with time[3],[4], so

$$k_{i,t}^{\infty,\mathrm{up}} = k^{\mathrm{fresh}}, \quad i = 1,\cdots,I,\ t = 1,\cdots,T.$$

A lower bound on k^∞ and an upper bound on R are derived from (5.8) and (5.7). Suppose it is known that a certain node i contains a fresh bundle, then $k_{i,1}^{\infty,\mathrm{lo}} = k^{\mathrm{fresh}}$. It then follows from the power peaking constraint (5.8) that, for this fresh bundle i and for time step $t = 1$,

$$R_{i,t}^{\mathrm{up}} = \frac{f^{\mathrm{lim}}}{k_{i,t}^{\infty,\mathrm{lo}} \sum\limits_{j=1}^{I} V_j}. \tag{5.15}$$

This implies that the burnup of this bundle during the first time step, given (when using forward discretization[5]) by (5.7), is bounded, giving a lower bound on $k_{i,t}^\infty$ where $t = 2$:

$$k_{i,t}^{\infty,\mathrm{lo}} = k_{i,t-1}^{\infty,\mathrm{lo}}(1 - \alpha\Delta_t R_{i,t-1}^{\mathrm{up}}). \tag{5.16}$$

This bound may become negative if $\alpha\Delta_t R_{i,t-1}^{\mathrm{up}} > 1$. Since we know that $k_{i,t}^\infty$ should be nonnegative, this sharpens the bound (5.15) on $R_{i,t}^{\mathrm{up}}$ to[6]:

$$R_{i,t}^{\mathrm{up}} = \min\left\{ \frac{f^{\mathrm{lim}}}{k_{i,t}^{\infty,\mathrm{lo}} \sum\limits_{j=1}^{I} V_j}, \frac{1}{\alpha\Delta_t} \right\}. \tag{5.17}$$

Bounds (5.17) and (5.16) can be computed iteratively for $t = 1,\cdots,T$. The EoC lower bound $k_{i,T}^{\infty,\mathrm{lo}}$ for a fresh bundle is also a BoC lower bound $k_{j,1}^{\infty,\mathrm{lo}}$ for any one year old bundle at node j, so that we can repeat this iterative process for bundle j. The process can be repeated in the same way until $R_{\tilde{\imath},t}^{\mathrm{up}}$ and $k_{\tilde{\imath},t}^{\infty,\mathrm{lo}}$, $t = 1,\cdots,T$ are computed for a node $\tilde{\imath}$ with a bundle of age L.

In our optimization model, it is not known in advance where the specific bundles are to be placed, so the numbers i, j and $\tilde{\imath}$ in the above computations can be chosen arbitrarily. Since the

---

[3] Although clear for physical reasons, this statement can be proved mathematically. See Appendix B, Section B.2 for details.

[4] For the BP model, this statement holds for $\overline{k}^\infty$, not for k^∞.

[5] When using central discretization, the Δt in (5.16) and (5.17) can be replaced by $\frac{1}{2}\Delta_t$.

[6] It is assumed that Δ_t is small enough to ensure stability. For the basic model this is already true when only three or four time steps per cycle are used; the situation for the BP model is discussed in Section 5.2.6.

lower bounds $k^{\infty,\mathrm{lo}}$ are monotonically decreasing during the iterative process, and consequently the upper bounds R^{up} are monotonically increasing, the bounds for node $\bar{\imath}$ (the node with a bundle of age L) are valid for all the nodes $i = 1,\cdots,I$. Hence we can apply the following bounds:

$$\begin{aligned} R^{\mathrm{up}}_{i,t} &= R^{\mathrm{up}}_{\bar{\imath},t}, \quad i=1,\cdots,I,\ t=1,\cdots,T, \\ k^{\infty,\mathrm{lo}}_{i,t} &= k^{\infty,\mathrm{lo}}_{\bar{\imath},t}, \quad i=1,\cdots,I,\ t=1,\cdots,T. \end{aligned}$$

Bounding k^{eff}

In order to derive upper and lower bounds for k^{eff}, we take a closer look at (5.5). In matrix form this is written as

$$k^{\mathrm{eff}}_t R_{\cdot,t} = GK_t R_{\cdot,t}, \tag{5.18}$$

where

$$K_t = \begin{pmatrix} k^{\infty}_{1,t} & & 0 \\ & \ddots & \\ 0 & & k^{\infty}_{I,t} \end{pmatrix}.$$

Equation (5.18) is an eigenvalue equation, and following the remarks after Equation (2.7) in Section 2.1.3, we are interested in the maximum eigenvalue k^{eff}_t and its corresponding positive eigenvector $R_{\cdot,t}$. Since GK_t is an irreducible nonnegative matrix, the maximum eigenvalue is strictly positive and has a strictly positive, up to a scalar unique, eigenvector (see Appendix B, Section B.1.1). We know that[7]

$$GK^{\mathrm{lo}}_t \leq GK_t \leq GK^{\mathrm{up}}_t. \tag{5.19}$$

Now the following well-known result may be applied.

Lemma 5.1 *Let A and B be two nonnegative irreducible $n \times n$ matrices. Let $\lambda(A)$ and $\lambda(B)$ be their maximum eigenvalues. Then*

$$A \leq B \Rightarrow \lambda(A) \leq \lambda(B).$$

A proof, which is given for a slightly more general case, is found in [7, Th.1.7.1]. Let $\lambda(GK^{\mathrm{lo}}_t)$ and $\lambda(GK^{\mathrm{up}}_t)$ be the maximum eigenvalues of the corresponding matrices, then it follows from (5.19) and Lemma 5.1 that

$$\begin{aligned} k^{\mathrm{eff,lo}}_t &= \lambda(GK^{\mathrm{lo}}_t) \quad t=1,\cdots,T, \\ k^{\mathrm{eff,up}}_t &= \lambda(GK^{\mathrm{up}}_t) \quad t=1,\cdots,T. \end{aligned}$$

Thus far, we have obtained upper and lower bounds for all variables except a lower bound on R. Though we know that all elements of R are strictly positive (which follows from the fact that GK_t is irreducible nonnegative), we could not guarantee a lower bound on the values of R. These lower bounds are therefore set to zero.

---

[7] Inequalities between matrices denote elementwise inequalities.

5.2.2 Addition of constraints

Besides tightening the bounds, convex constraints may be added in order to define a tighter polytope containing the nonconvex feasible area. Using the matrices K_t as defined in the previous section, it is known from (5.7) that $K_{t+1} < K_t$ elementwise. Hence Lemma 5.1 can be used to show that

$$k^{\text{eff}}_{t+1} \leq k^{\text{eff}}_{t},\ t = 1, \cdots, T-1. \tag{5.20}$$

Although this makes the model larger, explicit addition of these constraints helps to find feasible solutions with some nonlinear optimization packages.

5.2.3 Constraint relaxation

In the previous two subsections we were concerned about tightening bounds and addition of constraints. In the current subsection, relaxations of constraints are considered. The paradox between these two issues can be explained using Figure 5.3. In that figure, the feasible area is just a tiny nonconvex curve. This curve is described by nonlinear equations, and is intersected with the linear inequalities and one nonlinear nonconvex inequality. Regarding the linear inequalities, the optimization is helped if these bounds enclose the feasible curve as tight as possible. The nonlinear equalities describing the feasible area as a tiny curve make it very difficult for a solver to keep staying along this curve during optimization. Relaxing the nonlinear equality constraints may relieve this problem since the solver now can use feasible points in the shaded area during the optimization.

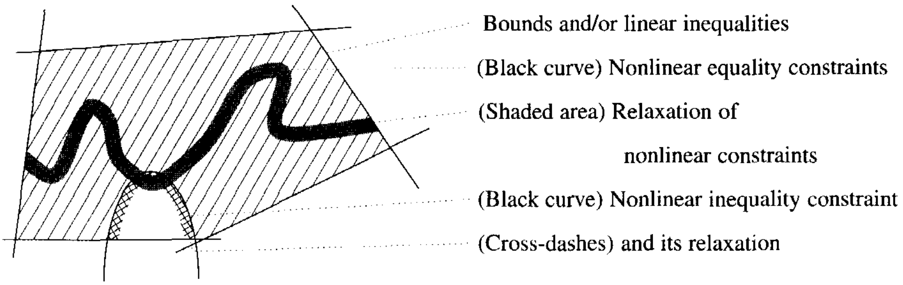

Bounds and/or linear inequalities
(Black curve) Nonlinear equality constraints
(Shaded area) Relaxation of nonlinear constraints
(Black curve) Nonlinear inequality constraint
(Cross-dashes) and its relaxation

FIGURE 5.3: Optimization is helped with tightening linear bounds and relaxation of nonconvex constraints. The feasible area is the curve determined by the nonlinear equality constraints, intersected with the dashed area that is determined by the linear and nonlinear inequality constraints. The nonlinear inequality constraint in this example disconnects the feasible area. Relaxing the nonlinear equalities makes the shaded area feasible. Relaxing the nonlinear inequality constraint makes the cross-dashed area feasible. Both relaxations together make the feasible area connected.

The nonlinear inequality constraint in the example disconnects the feasible curve, thereby disconnecting the feasible area. This makes it very difficult for a solver to reach an optimum in one part of the feasible area, if its starting point is in the other part. Even after relaxation of the nonlinear equality constraints the shaded area remains disconnected. This problem is relieved by relaxing the nonlinear inequality constraint, so that the cross-dashed area also becomes feasible. Since there is a nonempty intersection of the cross-dashed area and the shaded area, the feasible area is now connected.

In the sequel of this section, we discuss both types of relaxation for the fuel management optimization problem.

Power peak constraint relaxation

In a search for an optimal solution, it is more important to get at least some reasonable good pattern, than to get immediately a pattern that satisfies the power peaking constraint. Since the physical variables are continuous functions of the assignment matrix, the feasible area is nicely connected if the power peaking constraints are ignored. The power peaking constraints cut off some patterns, and it may happen that they disconnect the feasible area, just as the nonlinear inequality constraint in Figure 5.3. For these reasons it is attractive to relax the power peaking constraints. This was done by addition of a nonnegative variable ε to (5.8):

$$k^\infty_{i,t} R_{i,t} \leq \frac{f^{\text{lim}}}{\sum_{j=1}^{I} V_j} (1+\varepsilon). \tag{5.21}$$

During the optimization, ε is driven to zero by penalizing it in the objective function:

$$\text{maximize} \ \ k^{\text{eff}}_T - \theta\varepsilon,$$

where θ is a suitably large penalty parameter. The variable ε can be bounded above. For some very bad loading patterns, we obtained power peaks that were about four or five times as large as the maximum allowed power peak, and if we would hold all possible loading patterns into our feasible area, the bound on ε should be that large. In practice, it is sufficient to make the bound much smaller, since patterns that are in the vicinity of good loading patterns have much smaller power peaks. The precise value should be found by some experimentation; we found that values of $\varepsilon^{\text{up}} \in [.01; 1.0]$ lead in general to satisfying results.

It should be noted that the bounds R^{up} and $k^{\infty,\text{lo}}$ as derived in Section 5.2.1 are based on the maximum allowed power peak. If we use power peak relaxation using a nonzero ε^{up}, the bounds may cut off solutions that are feasible with respect to the relaxed power peaking constraints. This can be avoided by using (5.21) with $\varepsilon = \varepsilon^{\text{up}}$ in the bound calculations (5.17).

Turning equalities into inequalities

Another way of constraint relaxation is turning equality constraints into inequalities. This is in some sense equivalent to the creation of the shaded band around the feasible area in Figure 5.3. Care should be taken in determining the direction of these inequalities. The maximization of k^{eff}_T suggests that we can replace the kernel equations (5.5) with inequality constraints

$$k^{\text{eff}}_t R_{i,t} \leq \sum_{j=1}^{I} G_{i,j} k^\infty_{j,t} R_{j,t}, \ \ i = 1, \cdots, I, \ t = 1, \cdots, T$$

However, this is likely to introduce solutions where $R_{\cdot,t}$ is not a combination of eigenvectors, so that solutions without physical meaning are introduced.

The use of such equality relaxations depends on the solution method. The GRG method for example is an equality based method. If equalities are turned into inequalities, the solver converts them back into equalities by using slack variables. Other methods however, for example Projected Lagrangian based methods, might profit from this type of relaxation, since it gives the solver more freedom when choosing search directions. Since we used a GRG solver in most of our computations, we did not consider further use of equality relaxation.

5.2.4 Engineering constraints

Besides bounds tightening and constraint relaxation, the efficiency of nonlinear optimization can also be improved by adding engineering constraints. Engineering constraints reduce the feasible area in a heuristic way using expert knowledge of the problem. These constraints are imposed on the assignment variables, and the purpose is to cut off bad regions of the feasible area. In this way, the optimization algorithm is prevented to get stuck at local optima in such a bad region. Commonly accepted rules within the fuel management optimization community are for example (see also Section 3.3.2):

- No four fresh bundles in a square, since this leads either to too large power peaks, or to bad objective function values. This constraint should be implemented with great care: we encountered several very good solutions in reactor cores of medium size, containing such fresh squares somewhere in between the core center and the core periphery.
  In order to implement this engineering constraint, let $Q = \{(i^1,i^2,i^3,i^4) : (i^1,i^2,i^3,i^4)$ is a square$\}$. The constraints are then formulated as follows:

  $$\sum_{j=1}^{4}\sum_{m=1}^{M} x_{i^j,1,m} \leq 3, \;\; \forall (i^1,i^2,i^3,i^4) \in Q. \tag{5.22}$$

  Due to octant symmetry, some of these constraints can be tightened. On the boundary of the octant it may happen (after suitable reordering) that $i^1 = i^3$, $i^2 = i^4$ and $i^1 \neq i^2$. In this case, (5.22) reduces to

  $$2\sum_{j=1}^{2}\sum_{m=1}^{M} x_{i^j,1,m} \leq 3. \tag{5.23}$$

  This constraint may be divided by 2, giving a right hand side of 3/2 and a left hand side that is still integer for integer solutions. Thus we may replace the right hand side 3/2 by $\lfloor 3/2 \rfloor = 1$, resulting in

  $$\sum_{j=1}^{2}\sum_{m=1}^{M} x_{i^j,1,m} \leq 1.$$

  This constraint cuts off more non-integer solutions than (5.23). Another possible situation is that i^1, i^2 and i^3 are different, and $i^3 = i^4$. In that case, we add the redundant constraint

  $$\sum_{j=1}^{2}\sum_{m=1}^{M} x_{i^j,1,m} \leq 2$$

to (5.22) and divide again by 2, leading to the inequality

$$\sum_{j=1}^{3}\sum_{m=1}^{M} x_{ij,1,m} \leq \lfloor \frac{5}{2} \rfloor = 2.$$

- No fresh bundles in the center nodes. This constraint is especially applicable in the layouts of our test cores. Due to the assumption of octant symmetry and the existence of four center nodes, a fresh bundle in a center node immediately leads to a square of fresh bundles, and due to our restriction that two neighboring diagonal nodes contain the same bundles, the four corner nodes around the square also contain fresh nodes (*cf.* Figure 5.4). Such a configuration in the center is likely to cause too high power peaks.

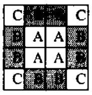


FIGURE 5.4: Center of an octant symmetric core. Nodes A and C contain the same bundle. If A is a fresh bundle, then so is C.

- No fresh bundles at the boundary of the core. Fresh bundles at boundary positions are not likely to be effective since lots of neutrons are spoiled in the water surrounding the core.
- No squares of the oldest bundles in the inner region of the core, since this will create 'holes' in the flux distribution.

In general, severe testing is required before an engineering constraint is applied, in order to avoid the exclusion of very good patterns, or even exclusion of the globally optimal pattern.

5.2.5 Reducing nonconvexity by reformulations

Reformulation of the model equations may lead to reduction of the nonconvexity. In this section, some of these reformulations are discussed.

Alternative reloading equation

Our model has constraints with linear and bilinear terms, and only the reloading equation (5.4) contains trilinear terms. An alternative formulation of (5.4) can be derived that involves only bilinear terms:

$$\sum_{i=1}^{I} x_{i,\ell,m} k_{i,1}^{\infty} = \begin{cases} \sum_{j=1}^{I} x_{j,\ell-1,m} k_{j,T}^{\infty}, & \ell = 2,\cdots,L,\ m = 1,\cdots,M, \\ k^{\text{fresh}}, & \ell = 1,\ m = 1,\cdots,M. \end{cases} \tag{5.24}$$

The left hand side of this equation selects the position i of the bundle of age ℓ and trajectory m. The right hand side selects the position j where this bundle was at the previous cycle. This constraint only contains bilinear terms, and the total number of non-zeros in the constraint gradient matrix is of order I^2 instead of the order I^3. For integer solutions, both formulations (5.4) and (5.24) are equal. For non-integer assignments however, the two methods are incompatible. There are situations in which (5.24) does not give any solution for the physical constraints, for example the case where both a bundle and its successor are evenly distributed over two different nodes:

$$x_{i,\ell,m} = x_{i,\ell+1,m} = x_{j,\ell,m} = x_{j,\ell+1,m} = \tfrac{1}{2}, \quad \text{for some } i, \ \ j \neq i, \ \ \ell < L, \ \ m.$$

Suppose $\ell = 1$, then the two reload equations for (ℓ, m) and $(\ell+1, m)$ according to (5.24) for this assignment read:

$$\begin{aligned} \tfrac{1}{2}(k^{\infty}_{i,1} + k^{\infty}_{j,1}) &= k^{\text{fresh}} \\ \tfrac{1}{2}(k^{\infty}_{i,1} + k^{\infty}_{j,1}) &= \tfrac{1}{2}(k^{\infty}_{i,T} + k^{\infty}_{j,T}). \end{aligned}$$

Since $k^{\infty}_{i,T} \leq k^{\text{fresh}}$ and $k^{\infty}_{j,T} \leq k^{\text{fresh}}$, this would imply that $k^{\infty}_{i,T} = k^{\infty}_{j,T} = k^{\text{fresh}}$, hence $R_{i,t} = R_{j,t} = 0$, $t = 1, \cdots, T-1$, which is impossible. On the other hand, since (5.24) compares only averages of infinite multiplication factors over nodes, there are assignments where non-realistic solutions are introduced with $k^{\infty}_{i,1} = k^{\text{fresh}}$ for almost all nodes i, while only for two or three nodes, $k^{\infty}_{i,1} \ll 1$. These solutions have very large but completely unrealistic objective values, which make the optimal solution value of the NLP senseless.

Addition of variables

A more standard way to remove trilinear terms is the introduction of extra variables. We recall the reloading equations (5.4):

$$k^{\infty}_{i,1} = \sum_{m=1}^{M} x_{i,1,m} k^{\text{fresh}} + \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} V_j x_{j,\ell-1,m} k^{\infty}_{j,T}, \quad i = 1, \cdots, I.$$

Now a new type of variable is introduced:

$k^{\text{EoC}}_{\ell,m}$: the EoC infinite multiplication factor of the bundle of age ℓ in trajectory m.

This variable is related to $k^{\infty}_{j,t}$ in the following way:

$$k^{\text{EoC}}_{\ell,m} = \sum_{j=1}^{I} x_{j,\ell,m} k^{\infty}_{j,T}, \quad \ell = 1, \cdots, L-1, \ m = 1, \cdots, M. \tag{5.25}$$

Using (5.25) as an extra set of constraints, we can replace the reloading constraints (5.4) with

$$\begin{aligned} k^{\infty}_{i,1} = {} & \sum_{m=1}^{M} x_{i,1,m} k^{\text{fresh}} \\ & + \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} k^{\text{EoC}}_{\ell-1,m}, \quad i = 1, \cdots, I. \end{aligned} \tag{5.26}$$

The use of (5.25) and (5.26) reduces the total number of nonlinear terms and the total number of nonzeros in the constraint gradient matrix. The practical results with this reformulation depend on the solver used. In our case, it leads to a significant decrease of the number of nonzeros in the constraint gradient matrix (see also Table 6.4), and it gives in general a reduction of the computation times with any solver. On the other hand, it removes the direct relations between the $k^{\infty}_{i,1}$ and $k^{\infty}_{j,T}$ which makes the problem less transparent, and makes the role of gradient information less clear. Practical results indicate a decrease of solution quality in different solvers.

5.2.6 Time-discretization

The optimization models make use of a discretization both in space and time. In space, nodal discretization is used, which is already discussed in Section 2.1.4. Regarding discretization in time, it is already noted that the two straightforward possibilities to discretize the burnup equations are forward and central discretization. Here both possibilities are worked out and compared. Remarks are also made about how many points in time are needed in order to get an accurate model, and where to place these points. Figure 5.5, presents some computational results with some test data on the basic model, with both forward and central discretization for various numbers of time steps. The time steps are equally distributed over the whole cycle. It is clear from the results that central discretization gives more accurate answers with much fewer discretization points.

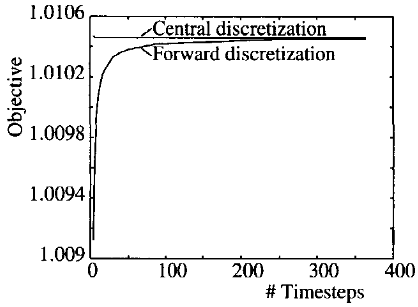


FIGURE 5.5: Central vs. forward differences solutions as function of the number of time steps. The central discretization solution is monotoneously decreasing in the number of time steps, the forward discretization solution is monotoneously increasing in the number of timesteps.

This observation still tells nothing about which discretization is better to use in our optimization system. A less accurate model may be perfectly suited for optimization purposes, provided that the *ranking* of all loading patterns with respect to the objective value remains the same. For a number of arbitrary loading patterns, we computed the objective value k^{eff}_T of the basic model both with forward and central discretization. The results are shown in Figure 5.6. Using 15 time steps, the maximum deviation between the forward and central discretization solutions is 0.2%. Using only 6 time steps, the maximum deviation is still only 0.5%. In view of the fact that our data are given in three, or at most four digits, we conclude that the forward model with

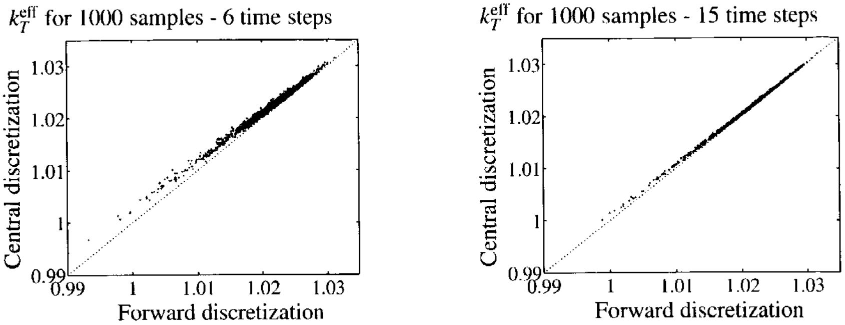


FIGURE 5.6: Comparison of the k_T^{eff} obtained by forward and central discretization for 1000 random loading patterns using $T = 6$ (left) and $T = 15$ (right). The time steps are equally distributed over the cycle length. Test data are taken from model Medium 1 (see Section 6.1).

6 time steps is very reasonable in order to identify a solution that is close to optimal. In most of the test runs that we performed on the standard model, we used this number of time steps.

In the case that burnable poisons are used, we have to take into account the stability of the solution. The (forward) discretized burnup equation for the BPs (2.27) is given as

$$\Sigma^{\text{p}}_{\text{a},i,t+1} - \Sigma^{\text{p}}_{\text{a},i,t} = -\alpha^{\text{p}}\Sigma^{\text{p}}_{\text{a},i,t}R_{i,t}\Delta_t$$

where R and $\Sigma^{\text{p}}_{\text{a}}$ are the variables (the latter must not to be confused with summation signs!). In order to achieve at least a stable solution, it is required that the value of $\Sigma^{\text{p}}_{\text{a},i,t+1}$ is nonnegative. This is possible only if

$$1 - \alpha^{\text{p}}R_{i,t}\Delta_t \geq 0$$

for each node i, which means that

$$\Delta_t \leq \frac{1}{\alpha^{\text{p}}R_{i,t}}.$$

To get an estimate for the maximum allowed Δ_t in advance, we need to estimate the maximal $R_{i,t}$. Using the power peak constraint (2.33) and assuming that power peaks will not be violated at nodes where $k^{\infty}_{i,t} < 1$, an estimate for feasible patterns is obtained:

$$R_{i,t} \leq \frac{f^{\text{lim}}}{\sum\limits_{i=1}^{I} V_i}.$$

Experiments showed that for bad patterns, the power peak can be four or five times the maximum allowed power peak, especially for larger cores[8]. So, to get stable solutions for all possible

---

[8] We could even construct pathological cases where the power peak is about a factor 7 above the maximum allowed power peak, but these examples are so obviously bad, also in terms of the objective function, that it is safe to assume that no optimization algorithm will ever reach such a case, except when the starting point is especially constructed to obtain this behavior.

patterns, we have the bound

$$\Delta_t \leq \frac{1}{5\alpha^{\mathrm{p}}} \cdot \frac{\sum_{i=1}^{I} V_i}{f^{\mathrm{lim}}}. \tag{5.27}$$

The highest power peak is attained at the beginning of a cycle, or, when BPs are added, slightly thereafter, so the changes in the BP concentration are largest near BoC. Therefore, the time intervals towards EoC may be longer than the maximum obtained in (5.27). We implemented an uneven distribution of time steps in which the last time step is the triple of the first time step:

$$\Delta_t = \frac{\text{Total cycle length}}{T-1}\left(\frac{1}{2}+\frac{t-1}{T-2}\right). \tag{5.28}$$

In this way, the number of time steps is limited, while maintaining stability in the parts with the largest power peaks.

Stability is easier to achieve when using central discretization. The burnup relation is then given by

$$\Sigma^{\mathrm{p}}_{\mathrm{a},i,t+1} - \Sigma^{\mathrm{p}}_{\mathrm{a},i,t} = -\alpha^{\mathrm{p}} \frac{\Sigma^{\mathrm{p}}_{\mathrm{a},i,t+1} R_{i,t+1} + \Sigma^{\mathrm{p}}_{\mathrm{a},i,t} R_{i,t}}{2} \Delta_t$$

which is equivalent to

$$\Sigma^{\mathrm{p}}_{\mathrm{a},i,t+1} = \frac{1-\alpha^{\mathrm{p}} R_{i,t} \frac{\Delta_t}{2}}{1+\alpha^{\mathrm{p}} R_{i,t+1} \frac{\Delta_t}{2}} \Sigma^{\mathrm{p}}_{\mathrm{a},i,t}. \tag{5.29}$$

This implies that, due to the division of Δ_t in the numerator by two, Δ_t may be twice as large as when using forward discretization in order to achieve stability.

5.3 The integer part

In this section, we focus on techniques to derive integer solutions. We start from the situation in which we have a local optimum of a continuous nonlinear optimization algorithm. Also, it is assumed that there exists an algorithm that calculates the physical variables, the objective function and the power peaks for a fixed assignment. Using these tools we derive integer solutions using the techniques that are discussed in Section 4.3, that will be worked out here for our specific situation. We first discuss a simple rounding heuristic. The second technique is Outer Approximation. We have not implemented this technique from scratch, but used the software package DICOPT. The third method is Branch-and-Bound. Only a restricted version of this algorithm was implemented for a simplified version of the reactor problem. Based on this experience, problem-related implementation issues for a larger-scale implementation are discussed. The fourth technique is a combination with neighborhood search. Besides these four techniques, we also experimented with generation and addition of cuts. We used cuts not only with the purpose of getting integer solutions, but the main purpose in our application was to cut off bad local optimal solutions, so they combine improvement of the NLP solutions and improvement of the integer solutions. Therefore the generation and the use of cuts is discussed separately in Section 5.4.

5.3.1 Rounding heuristics

Due to the nature of our basic model, many integer solutions turn out to be local optima. In several cases, the NLP solver terminates immediately with an integer value. When the returned optimum is not integer, this is often due to a power peaking limit (see Figure 5.7), and the number of nonintegers is usually small[9]. In order to quickly find an integer solution, we start with the relaxed solution and enumerate all possible neighboring integer solutions such that

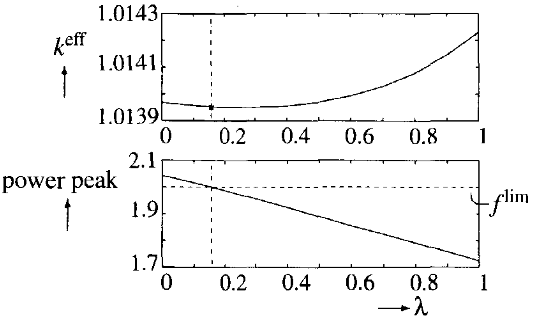


FIGURE 5.7: Fractional solution due to the power peak constraint. The values $\lambda = 0$ and $\lambda = 1$ represent two integer solutions x^0 and x^1, other values represent fractional solutions $(1-\lambda)x^0 + \lambda x^1$. The k^{eff}-plot shows that $x = 0$ and $x = 1$ are local maxima. The power peak plot shows that in $x = 0$ the power peak f^{lim} is exceeded. Optimization of the continuous relaxation gave the local optimum indicated at $x \approx 0.16$. Rounding leads to the (in this case even better) solution $x = 1$.

- ones in the relaxed solution remain ones, and
- zeros in the relaxed solution remain zeros.

Suppose we have the following part of the assignment matrix:

$$\begin{pmatrix} 0.2 & 0.8 & & & \\ 0.8 & 0.2 & & & \\ & & & 0.5 & 0.5 \\ & & 0.7 & & 0.3 \\ & & 0.3 & 0.5 & 0.2 \end{pmatrix}$$

(zeros in this matrix are left blank for simplicity). There are two possibilities to create integer solutions in the first two rows and columns, and independently there are three binary solutions in the last three rows and columns, so there are six possible binary solutions in total.

The feasible rounding with best objective value is returned, or, if none of the rounded solutions is feasible, the best infeasible rounding is returned.

---

[9] This is not true for the model of Section 2.3.3. Still, the described procedure is applicable as part of a more general rounding scheme.

5.3.2 Outer approximation

This method, discussed in Section 4.3.3, is implemented in the software package DICOPT [37], running within the high-level modeling language GAMS [13]. The algorithm solves a sequence of nonlinear programming problems and mixed-integer linear programming problems. Let y denote the physical variables, and x denote the binary variables, then the DICOPT algorithm, as described in [37], is listed in Algorithm 5.3. If the first relaxed solution happens to be integral, then the algorithm is terminated. If not, a linearization is made around the current solution point and added to the mixed-integer linear optimization (MILP) problem, containing the union of all linearizations of iteration 0 to i. The MILP problem is solved and an integer solution is returned, which is a loading pattern in our case. The physical variables are not accurate due to the linearization. Therefore, the NLP problem is solved. From the resulting point in the space of all variables, a new linearization of the constraints is made and added to the MILP subproblem, until the objective function of the NLP problem starts worsening.

DICOPT

```
begin
  read $(x^0, y^0)$;  {read the user-supplied starting guess}
  Solve the relaxed (NLP) problem $\rightarrow$ $(x^1, y^1)$.
  if $x^1$ is integer then stop: Optimal solution found;
  else
    $i := 1$;
    $z := -\infty$;  {Initialize lower bound on maximization}
    repeat
      $\bar{z} := z$;
      Linearize around $(x^i, y^i)$ and
        add constraints to the linearized (MILP) problem;
      Add integer cuts to the MILP problem;
      Solve the MILP problem to the integer optimal solution $(x^{i+1}, y^{i+1})$;
      Solve the NLP problem with $x = x^{i+1}$ being fixed.
      Let the solution be $(x^{i+1}, y^{i+1})$ and the optimum value be $z$;
      $i := i+1$;
    until $(\bar{z} \geq z)$;
  endif
end.
```

ALGORITHM 5.3: DICOPT algorithm to solve nonlinear mixed-integer problems.

With the basic reload pattern optimization model, the only control variables are the integer assignment variables, which are fixed in all NLP subproblems, except the first one. These NLP subproblems therefore reduce to straightforward computations of the physical variables for a fixed loading pattern and the only real optimization problems are the MILP subproblems. In this sense, the algorithmic structure somewhat resembles the Sequential Linear Programming (SLP) method. A major difference with this method is the optimization of the first relaxed NLP problem, that may be solved with any NLP algorithm.

In the model where continuous BP optimization is included, the correspondence with SLP does no longer hold. In this case, the BP concentration is a continuous control variable, that is optimized in the NLP subproblems.

Different solvers can be attached to DICOPT to solve the NLP subproblems and the MILP subproblems. For the solution of the NLP subproblems, we used the nonlinear solver CONOPT [27], which gave more robust solutions than the competing solver MINOS5. It makes use of the Generalized Reduced Gradient method [45]. The MILP-solutions are obtained by the solver CPLEX, one of the most advanced MILP-solvers today. It internally uses the linear simplex method combined with a Branch-and-Bound approach to reach an integer solution[10]. Branch-and-Bound methods have the property that the total solution time is unpredictable in advance. Still the average behavior of the algorithm can be influenced by changing some Branch-and-Bound parameters. After testing different parameter settings, we concluded that the default node selection strategy in CPLEX 6.0 was not optimal to our problem. The standard strategy is to choose that node from the node table with the best objective function of the associated relaxed problem (best-bound strategy). If there are multiple optimal and near-optimal solutions in different branches, this may cause much branching rather high in the Branch-and-Bound tree. If, on the contrary, the depth first node selection strategy is chosen, integer solutions are found quickly. Typically it still needs some branching to prove that the solution found is within optimality tolerances. We observed slight improvement of the algorithm when the depth first strategy was selected.

5.3.3 *Branch-and-Bound*

Another way to find integer solutions is to implement a nonlinear Branch-and-Bound algorithm. The basic structure is the same as with the linear Branch-and-Bound algorithm as described in Section 4.3.2. The main difference is that we cannot guarantee that the bounds found by solving the relaxed problems are correct *i.e.*, it may happen that a relaxed problem returns a local optimal solution instead of a global one. Such a node might be fathomed while it in fact may contain the global optimal solution. In this way, Branch-and-Bound reduces to a heuristic method, and a straightforward implementation leads to long computation times and bad local optimal solutions. Fortunately, there are several possibilities for improvement.

The risk of throwing away potentially good nodes is reduced by keeping nodes in the node table that are worse than the cutoff value by less than a given fraction. This fraction depends on the level of the node: for nodes high in the tree more freedom is allowed than for lower-level nodes. As a consequence, the optimal solution found may become better, but computation time also increases. It is therefore still important to find good quality bounds in the different nodes, and to find meaningful cutoff values. This can be obtained by issuing a depth-first search, that is likely to lead relatively fast to integer solutions. Another possibility is to apply some rounding heuristic in the different nodes, to force an integer solution value. If this integer solution is worse than the fractional solution obtained in the node, then the node should be kept in the node table, since the continuous optimum suggests that better solutions are possible. Otherwise the integer solution value may be compared to the best known solution value. Heuristic rules are

---

[10] Note that this linear Branch-and-Bound method, that is hidden in the CPLEX solver, is different from the problem-specific nonlinear Branch-and-Bound method that is discussed in Section 5.3.3.

then needed to decide when a node will be fathomed. Additional research and computational experience is needed to find suitable rules.

With the different adaptations we tested thus far with the basic model, either the quality of the results is poor, or the algorithm requires an astronomical amount of time to find reasonably good solutions. This is due to the fact that there are very many local optima that have comparable objective values, but that are not close to each other in the search space. Within a node, the local optimum found in each node depends on the specific starting value that is used to start the NLP algorithm in that node; therefore a balanced starting value selection strategy for all nodes has to be developed. These starting values must be generated by applying the same rules that are used for the starting value of the relaxed NLP problem, *i.e.*, the bundles that are not yet fixed must have fractional values.

Due to the nonconvexity structure of the several models, it may be expected that the Branch-and-Bound method works better for the new model as developed in Section 2.3.3. This model still has many local optima, but at least the property that many integer solutions are local optima is lost, and the objective function has a much smoother shape.

Special attention in the Branch-and-Bound has to be paid to the power peaking constraints, since these constraints may divide the feasible region into different disconnected areas. It is probably helpful to relax the power peaking constraints completely in the top level nodes, and to make them gradually tighter in lower level nodes. In the top nodes, we are only interested in the possible values of the objective function in the different nodes, rather than in precise feasibility. The lower in the tree, the more we work to find real loading patterns, that satisfy the power peaking constraints.

A different way to speed up the Branch-and-Bound algorithm is the use of a much coarser model high in the tree. In the top nodes we may for example use a coarse model with a few discretization points, that at least gives an indication of what objective function values we may expect in the different nodes. When branching down in the tree, a more precise model may be used. This is possible since we found that the precision of coarse models (within some limits) is good enough to give a rather precise distinction between good and bad loading patterns (*cf.* Figure 5.6)

It is important to have a dedicated Branching strategy such that most bad patterns are cut off as high in the tree as possible. Therefore the algorithm should branch first on the bundles or positions that have most influence on the objective function. These are the center positions, the boundary positions, the fresh bundles and the oldest bundles. Middle-aged bundles, and core positions between the core center and the periphery should be fixed only rather deep in the tree.

The Branch-and-Bound algorithm may be extended with a cut-generating algorithm. Instead of dividing the problem into two subproblems with a variable fixed to 0 and 1, respectively, the problem can be divided into two subproblems that are separated by a suitable cut. Currently, we have an implementation of a cut-generating algorithm, that is described in Section 5.4, in which only one side of a cut is explored. Due to the nonconvexity of the problem, this is a heuristic algorithm, since a cut may wrongly cut off the global optimum. When both sides of a cut are explored, such a situation may be avoided.

5.3.4 Pairwise interchange

One of the aims of this project is to compare the MINLP approach with a direct search approach. Our results from MINLP are compared to the results of a pairwise interchange (PI) algorithm. The basic pairwise interchange algorithm selects an initial parent pattern, explores all neighbors that can be found by exchanging two bundles, and selects the best one as the new parent. From this pattern, a new neighborhood search is launched, and the process is repeated until no improving neighbor can be found.

The results of a standard PI were improved by handling infeasible neighbor patterns in a special way. Neighbor patterns are infeasible when power peaking constraints are violated. Nevertheless, if they have a very good objective function value, they may have other neighbor solutions that also have a good objective function value and satisfy the power peaking constraints. For this reason we keep track of the best found infeasible solution that is better than the best known feasible solution. If the current parent pattern has no improving feasible neighbor solutions that can be chosen as new parent, the best known infeasible solution is selected as the new parent. If the search around this parent results in a better feasible or infeasible neighbor, this neighbor will be the new parent, otherwise the search is terminated. This PI algorithm is listed in Algorithm 5.4. The evaluation of the fixed loading patterns is done using the method described in Section 5.1.3.

The PI algorithm can be improved by various means. Instead of computing the complete core in each neighbor solution, one may use depletion perturbation theory, as is shown in [39]. Also the use of engineering constraints can speed up the algorithm, at the risk of loosing some good, possibly optimal solutions. In our test implementation, we have not used such constraints, primarily because the aim of our work is to explore the applicability of MINLP techniques and actually we have not implemented many engineering constraints into the MINLP method either.

5.3.5 Combination of methods

In many difficult optimization problems containing a combinatorial structure, the best results are found when combining several algorithms. An advantage of MINLP compared to direct search techniques is that it finds solutions of reasonable quality in a relatively short time. It can do larger steps at once, and based on gradient information it can step directly in an improving direction, without the need to explore a number of arbitrary patterns in a small neighborhood. On the other hand, PI is able to find improving neighbors that cannot be found by MINLP, if the objective function in the continuous optimization space has local optimal solutions in the integer solutions. Starting from an arbitrary solution, PI may need a large number of steps to reach a reasonably good solution. For these reasons we also have tested a combined strategy, in which we start with running a MINLP algorithm. After termination of this algorithm, the search is continued using PI. This way, we try to reduce the running time of PI, while on the other hand we improve the solutions obtained by MINLP. In many cases, the results are also better then the ones that were found by PI alone.

```
PI($x, f_x$)

get_starting_pattern($x$);
EvalFixed($x, f_x$);
if $x$ is feasible then
    $f_B := f_x$; $f_I := 0$; $x_B := x$; $inf\_explored := False$;
else
    $f_I := f_x$; $f_B := 0$; $x_I := x$; $inf\_explored := True$;
endif
$Stop := False$;
repeat
    $f_B^{old} := f_B$; $f_I^{old} := f_I$;
    for each neighbor $\hat{x}$ of $x$ do
        EvalFixed($\hat{x}, f_{\hat{x}}$);
        if $\hat{x}$ is feasible and $f_{\hat{x}} > f_B$ then
            $f_B := f_{\hat{x}}$; $x_B := \hat{x}$;
        elseif $f_{\hat{x}} > \max(f_B, f_I)$ then
            $f_I := f_{\hat{x}}$; $x_I := \hat{x}$; $inf\_explored := False$;
        endif;
    endfor;
    if $f_B > f_B^{old}$ then
        $x := x_B$;
    elseif $f_I > f_B^{old}$ and not $inf\_explored$ then
        $x := x_I$; $inf\_explored := True$;
    else
        $Stop := True$;
    endif
until $Stop := True$;
```

ALGORITHM 5.4: PI algorithm exploring infeasible neighbors. The parameter *inf_explored* is needed to avoid repeated selection of the same infeasible pattern as parent.

5.4 Cuts

Generating and adding cuts is a well-known technique to speed up the solution process of linear mixed-integer optimization problems. In our basic model, the most difficulties were not observed in obtaining integer solutions, since the local optimal solutions of the continuous relaxation often had only a few non-integer assignments. Simple rounding was amazingly effective in finding very reasonable integer solutions. The most difficult part of our model is the nonlinear part. Due to the nonconvexity structure, we often get stuck in local optimal solutions. In order to overcome this problem, a set of cuts were developed that are used with the purpose of cutting off bad local optimal solutions. This way, the NLP algorithm finds other, hopefully better, local optima. In this section we discuss the different cuts that we have implemented. In the discussion we concentrate on the basic model, although the cuts are also applicable to the BP model or the improved fractional model (Section 2.3.3).

The cuts that were implemented so far are based on smoothing the power shape in the core. Solutions where power peaking constraints are exceeded, should be avoided. Another motivation for power peak smoothing is that there is a weak negative correlation between power peak and k^{eff}, *cf.* Figure 5.8. The power power peak in a node i is described as

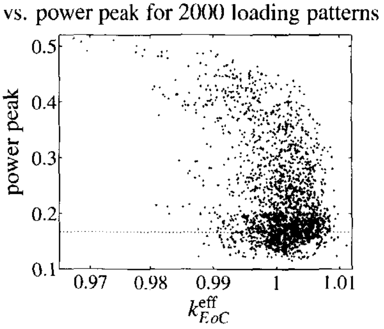


FIGURE 5.8: Results of a fixed core evaluation for 2000 patterns on test data set Small 1.

$$\max_t k^{\infty}_{i,t} R_{i,t},$$

since the total power summed over all nodes is constant in time. The power peak is decreased by reducing either $k^{\infty}_{i,t}$ or $R_{i,t}$. These two quantities are related by the kernel equation (5.5):

$$k^{\text{eff}}_t R_{i,t} = \sum_{j=1}^{I} G_{i,j} k^{\infty}_{j,t} R_{j,t}.$$

Since the matrix G is diagonally dominant, reducing $k^{\infty}_{i,t}$ for some i will in general reduce the corresponding $R_{i,t}$, although it may occasionally happen, due to normalization and influences in other parts of he core, that $R_{i,t}$ increases. The general idea of the cuts type 1, 2 and 3 that are described on the forthcoming pages is based on reduction of $k^{\infty}_{i,t}$ in nodes i where the power peak is too high. So, if node i has the maximum power peak in the current iteration p, the basic form of such a cut is:

$$k^{\infty}_{i,t} \leq \delta k^{\infty,p}_{i,t}, \tag{5.30}$$

for some suitable $\delta < 1$. Since in our model k^{∞} is a variable that depends indirectly on the control variables $x_{i,\ell,m}$, the cuts are applied to these control variables instead of $k^{\infty}_{i,t}$ directly. Here the property is used that k^{∞} decreases during the lifetime of the bundle, and that bundles of the same age have k^{∞}-values that are rather close. The restriction (5.30) is replaced by the restriction that the age of the bundle in node i and/or its neighbors should be above a given value.

The general algorithm with cuts is given in Algorithm 5.5. In each iteration we start by computing an initial pattern. Using this starting pattern, the continuous NLP is solved. If the solution is feasible, integer and better than the best found solution thus far, it is stored. If it is not feasible then cuts are derived from this solution. After this processing step, and if the solution is not integer, a number of integer solutions are derived from this solution. In our implementation this is done by simple rounding. These solutions are 'processed' in the same way as the NLP solution. Redundant cuts are then removed. This algorithm is repeated until some stopping criterion is met.

```
General cut-algorithm

repeat
    Find_initial_pattern($x$);
    NLP_solve($x, f_x$);
    Process_solution($x, f_x$);
    Derive a set $X$ of integer solutions from $x$;
    for each $\hat{x}$ in $X$ do
        EvalFixed($\hat{x}, f_{\hat{x}}$);
        Process_solution($\hat{x}, f_{\hat{x}}$);
    endfor
    Remove cuts that are redundant or timed-out.
until stopping criterion met;
```

```
Process_solution($x, f_x$)

if $x$ feasible and $x$ integer and $f_x > f^{\text{best}}$ then
    Store $x$ as $x^{\text{best}}$ and $f_x$ as $f^{\text{best}}$
if $x$ not feasible then
    Derive cuts from $x$.
```

ALGORITHM 5.5: General outline of the cut-algorithm.

5.4.1 Cut type 1

The power peak in some node i is mainly caused by the reactivity of itself and the adjacent nodes. High reactivity and a large flux in the surrounding nodes also causes a large flux in node i (see Figure 5.9). In type 1 cuts, we assume that the power peak in node i can be reduced by putting less active bundles in one or more of its neighbors, and it does not matter which one of the nodes this is. This constraint is implemented by computing the sum of the ages of the bundles in i and its surrounding nodes in the current solution. In the next iteration it is required that this sum should be larger than in the current solution, so that the bundles in the relevant nodes are older on average.

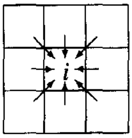


FIGURE 5.9: High fluxes in surrounding nodes influence the power in node i.

Let the assignment variables in the current iteration p be denoted as $x^p_{i,\ell,m}$, denoting the fraction of the bundle with age ℓ from trajectory m that is placed in node i at the pth iteration. Let i be the node on which the cut is posed, and let $\boldsymbol{N}(i)$ be the set of neighbor nodes including i

itself, consisting of all nodes as given in Figure 5.9. We compute

$$A_i^1 = \sum_{j \in N(i)} \sum_{\ell=1}^{L} \sum_{m=1}^{M} \ell x^p_{j,\ell,m} \tag{5.31}$$

and add the constraint

$$\sum_{j \in N(i)} \sum_{\ell=1}^{L} \sum_{m=1}^{M} \ell x_{j,\ell,m} \geq \lfloor A_i^1 + \delta^1 \rfloor \tag{5.32}$$

for some suitable δ^1 that is initially chosen greater than 1. We round to integer since in a fractional solution, A_i may be non-integer, but the solution we want to obtain should always be integer. The choice of δ^1 is in principle free. In our implementation it consists of two elements: an element that depends on the magnitude of the power peak violation at node i in iteration p, and a fixed factor:

$$\delta^1 = \alpha^1 \delta^{\mathrm{PP}} + \varepsilon^1 \tag{5.33}$$

where α^1 and ε^1 are constants, and

$$\delta^{\mathrm{PP}} = \max_t (k^\infty_{i,t} R_{i,t}) \, \frac{\sum_{j=1}^{I} V_j}{f^{\mathrm{lim}}}. \tag{5.34}$$

A value $\delta^{\mathrm{PP}} > 1$ corresponds to a violation of the power peaking constraint. The values of α^1 and ε^1 should be tuned by experiment. As an alternative to the definition (5.33) of δ^1 we also tested

$$\delta^1 = \alpha^1 \log(\delta^{\mathrm{PP}}) + \varepsilon^1. \tag{5.35}$$

This is motivated by the fact that the constraint can be too strong for large violations of the power peak. In such a case it may occur that no assignment can be made that satisfies all added cuts. This holds especially for smaller cores. In those cores, the power peak can be very high since placing one or two fresh bundles at center positions has major influence on the whole core (we actually encountered values $\delta^{\mathrm{PP}} > 5$). Also, these small cores have less flexibility to satisfy the cuts since there are less bundles of each age.

5.4.2 Cut type 2

In the type 1 cuts, it is implicitly assumed that the node i itself and its direct neighbors have the same influence on the power peak in i, whereas the influence of bundles that are further away is completely neglected. This assumption can be refined by taking into account the matrix $G_{i,j}$. As follows from the kernel equation, the contribution of the different nodes j to the flux value ϕ, hence to the removal rate R, and finally to the power peak in node i, is proportional to $G_{i,j}$:

$$R_{i,t} = \frac{1}{k_t^{\mathrm{eff}}} \sum_{j=1}^{I} G_{i,j} k^\infty_{j,t} R_{j,t}.$$

As in the type 1 cut, we assume that the power in node j is related to the age of the bundle, but instead of counting the ages of all neighbors as in (5.31), we count the ages with relative weights $G_{i,j}$:

$$A_i^2 = \sum_{j=1}^{I} G_{i,j} \sum_{\ell=1}^{L} \sum_{m=1}^{M} \ell x_{j,\ell,m}^{p} \tag{5.36}$$

and the cut is formulated as

$$\sum_{j=1}^{I} G_{i,j} \sum_{\ell=1}^{L} \sum_{m=1}^{M} \ell x_{j,\ell,m} \geq A_i^2 + \delta^2 \tag{5.37}$$

where δ^2 is defined in the same way as δ^1:

$$\delta^2 = \alpha^2 \delta^{\mathrm{PP}} + \varepsilon^2 \tag{5.38}$$

or, alternatively,

$$\delta^2 = \alpha^2 \log(\delta^{\mathrm{PP}}) + \varepsilon^2. \tag{5.39}$$

for suitable values of α^2 and ε^2. Note that the right hand side of (5.37) is not rounded to integer. Contrary to (5.32), the left hand side can be non-integer even when all x-variables are integer, thus non-integer right hand sides make sense. As can be seen from Figure 5.10, the relative importance of the neighbor nodes decreases about one order of magnitude for each step away from the center node i. The precise effect of the cut depends on the values of α^2 and ε^2, that should be found by experimenting, but in general this cut leaves more flexibility for the assignment variables than the type 1 cut. For some relatively small power peak violations it may be possible to increase the age of the center node, while the age of a neighbor node is decreased, which still has the net effect of a decreasing left hand side of (5.37).

.003	.007	.008	.007	.003
.007	.025	.050	.025	.007
.008	.050	.600	.050	.008
.007	.025	.050	.025	.007
.003	.007	.008	.007	.003

FIGURE 5.10: Probabilities $G_{i,j}$ that neutrons produced in node j are absorbed in the center node. The matrix G here is symmetric; note that in general the sum of probabilities need not to be one.

5.4.3 Cut type 3

The cut of type 2 may be refined further by taking into account the height of the power peak in node i and its neighbors. Suppose the power peak is exceeded in node i. If neighbor j_1 has a relatively large power, while neighbor j_2 with the same $G_{i,j}$-value has a much lower power, the bundle j_1 contributes more to the power peak in i than the bundle j_2. The power peak in i

might be suppressed by putting an older bundle in j_1, even while a less old bundle is placed in j_2. This is accomplished by making the cut also dependent on the power peak in each node j. Let

$$P_j^p = \max_t (k_{j,t}^{\infty,p} R_{j,t}^p)$$

be the power peak in node j in the current solution p, then define

$$A_{i,p}^3 = \sum_{j=1}^{I} G_{i,j} P_j^p \sum_{\ell=1}^{L} \sum_{m=1}^{M} \ell x_{j,\ell,m} \tag{5.40}$$

leading to the cut definition

$$\sum_{j=1}^{I} G_{i,j} P_j^p \sum_{\ell=1}^{L} \sum_{m=1}^{M} \ell x_{j,\ell,m} \geq A_{i,p}^3 + \delta^3 \tag{5.41}$$

where δ^3 may be defined as in the previous cut types:

$$\delta^3 = \alpha^3 \delta^{\mathrm{PP}} + \varepsilon^3 \tag{5.42}$$

or alternatively the logarithmic variant like (5.39). Note that we have added the p-subscript to in the definition of $A_{i,p}^3$. In the previous cuts, the coefficients in the left hand sides of the cuts (5.32), (5.37) were independent of p. So the cuts obtained in different iterations for the same node were linearly dependent and only either the strongest or the latest obtained cut had to be used. For the current type of cuts (5.41) this is no longer true, and if we use the cuts for more than one iteration, the different right hand sides of the subsequent iterations have to be saved separately.

This raises also the question how long the cuts should be kept in the solution process. The type 3 cut with its coefficients and right hand sides depending on the power peak in the current solution is very dependent on that specific solution. The power peak in the neighboring nodes also depends on nodes that are further away, and a high power peak in a neighboring node may be caused by a still higher power peak in a neighbor of this neighbor. The cut is therefore valid on a local scale only, that is on solutions that are not too different from the current solution. So, although the cut is useful in order to escape from the current solution, it should be dropped after one or two consecutive iterations.

5.4.4 Power peak strengthening.

The cut as described in this section is completely different from the three types of cuts described until now. It works directly via the power peaking constraint and is a restriction imposed on k^∞ and R instead of x. Recall the relaxed power peaking constraint (5.21):

$$k_{i,t}^{\infty} R_{i,t} \leq \frac{f^{\lim}}{\sum_{j=1}^{I} V_j} (1 + \varepsilon). \tag{5.43}$$

Instead of penalizing the power peak when it exceeds the limit value, we may already start penalizing at a lower value. If we know that the power peak is exceeded in node i in the current solution, we may penalize the power peak in this node at a lower value, in order to drive the optimization algorithm to a different solution. The power peaking constraint is to that end replaced by

$$k^{\infty}_{i,t} R_{i,t} \leq \frac{f^{\lim}}{\sum\limits_{j=1}^{I} V_j} \left(\frac{1}{\delta^4} + \varepsilon\right) \tag{5.44}$$

where

$$\delta^4 = \alpha^4 \delta^{\mathrm{PP}} + \varepsilon^4 \tag{5.45}$$

or the log-variant as (5.39).

5.4.5 Warm start after the addition of cuts

An important issue in all cut algorithms is the implementation of a good warm start, that is, the implementation of a starting point for the next NLP after the problem is changed. The current solution typically becomes infeasible when cuts are added. If we start with the current solution or quite close to it, the algorithm may get stuck in a feasible solution that is somewhat different from the previous solution in order to satisfy the cuts, but that is still in the attraction area of the previous local optimum. In order to overcome that limitation, we designed two possible alternative loading patterns. In our notation, we use the following starting patterns, where $\bar{x}$ is the matrix $x_{i,\ell,m}$, $i = 1, \cdots, I$, $\ell = 1, \cdots, L$, $m = 1, \cdots, M$:

1. $\bar{x}^{(1)}$: The current solution;
2. $\bar{x}^{(2)}$: The homogeneous core;
3. $\bar{x}^{(3)}$: The starting pattern obtained by model 1 as described in Section 5.1, with weight factor $\omega = 0$.
4. $\bar{x}^{(4)}$: As $\bar{x}^{(3)}$, but now with weight factor $\omega = 1$. Note that a pattern for $0 < \omega < 1$ can be computed as $\omega\bar{x}^{(4)} + (1-\omega)\bar{x}^{(3)}$.
5. $\bar{x}^{(5)}, \cdots, \bar{x}^{(4+P)}$: The P best (integer and non-integer) feasible patterns found until now, where P is a model parameter that is determined by experimentation.

Warm start model I

In this model, a pattern is found as linear combination of $\bar{x}^{(1)}$ to $\bar{x}^{(4+P)}$ that –if possible– satisfies the cuts. If this is not possible then the violation of the cuts is minimized. This is accomplished by solving the optimization problem:

$$\begin{array}{ll}\underset{\bar{x},\lambda,\zeta\geq 0}{\text{maximize}} & \sum\limits_{p=1}^{4+P}(\gamma_p\lambda_p)-M\cdot\sum\limits_{c=1}^{C}\zeta_c^2\\ \text{subject to} & \bar{x} = \sum\limits_{p=1}^{4+P}\lambda_p\bar{x}^{(p)}\\ & \sum\limits_{p=1}^{4+P}\lambda_p=1\\ & (\text{Value of cut } c)\geq(\text{RHS of cut } c)-\zeta_c,\quad c=1,\cdots,C\end{array}$$

where C is the number of cuts, the penalty parameter M is a big number, and γ_p, $p=1,\cdots,4+P$ are suitably chosen weights. As an alternative, one may use a sum of logarithms in the objective:

$$\underset{\bar{x},\lambda,\zeta\geq 0}{\text{maximize}}\sum_{p=1}^{4+P}(\gamma_p\log(1+\lambda_p))-M\cdot\sum_{c=1}^{C}\zeta_c^2,$$

resulting a more evenly distributed combination of the patterns $1,\cdots,4+P$. The model is a simple convex optimization problem with linear constraints, that can be solved in a fraction of a second. Without the logarithmic term, the problem reduces to a simple quadratic optimization (QP) problem.

Warm start model II

In warm start model I, the number of patterns from which can be chosen is small, namely all convex combinations of the patterns $1,\cdots,4+P$. Therefore it occurs quite frequently that the new solution is close to the old one. It may be more advantageous to have a pattern that strictly satisfies the cuts, but which is not necessarily a combination of the given patterns. In the model below the solution is forced to satisfy the cuts, and the objective is to minimize the (weighted) deviation of the starting pattern from the given patterns, using a quadratic penalty function.

$$\begin{array}{ll}\underset{x,\zeta\geq 0}{\text{minimize}} & \sum\limits_{p=1}^{4+P}\gamma_p\sum\limits_{i=1}^{I}\sum\limits_{\ell=1}^{L}\sum\limits_{m=1}^{M}(x_{i,\ell,m}-\bar{x}_{i,\ell,m}^{(p)})^2 \;-\; \sum\limits_{c=1}^{C}\log(\zeta_c+\varepsilon^{\log})\\ \text{s.t.} & \sum\limits_{i=1}^{I}V_i x_{i,\ell,m}=1,\quad \ell=1,\cdots,L,\; m=1,\cdots,M\\ & \sum\limits_{\ell=1}^{L}\sum\limits_{m=1}^{M}x_{i,\ell,m}=1,\quad i=1,\cdots,I\\ & (\text{Value of cut } c)\geq(\text{RHS of cut } c)+\zeta_c,\quad c=1,\cdots,C.\end{array}$$

This model is again a simple convex minimization problem. It gives no solution if and only if there is no solution satisfying all cuts. Depending on the values γ_p and $\varepsilon^{\log}$, the penalty term in the objective drives the solution away from the boundary of the feasible area. This is especially advantageous when the last added cuts do not cut off the attraction area of the last obtained local optimum completely and γ_1 is relatively large. in that case a model without this penalty term would very likely result in a solution at which the last added cut is active, and that is still in the attraction area of the last obtained local optimum. This would resulting in a new solution that is very close to the previous one, with an objective function that is worse. This was observed quite frequently in experiments without this penalty term.

The relative weights γ_p should be determined by experience. On the one hand, one can argue that the current solution is already a good local solution, and we want to stay close to it, so γ_1 should be relatively large. The same reasoning applies to $\gamma_5, \cdots, \gamma_{4+P}$. On the other hand, starting closer to $\bar{x}^{(4)}$, hence closer to the original starting point, enlarges the change to reach another, hopefully better, local optimum. The patterns $\bar{x}^{(2)}$ and $\bar{x}^{(3)}$ are usually bad, and the corresponding γ_2 and γ_3 should be small or zero.

In the case both $\bar{x}^{(1)}$ and $\bar{x}^{(4)}$ are feasible, we might like to have a 'centered' solution. This is not achieved with the objective above, since whenever $\gamma_1 > \gamma_4$ such a solution would always be equal to $\bar{x}^{(1)}$. To avoid this, one might switch to a centered solution by using an interior point method.

5.4.6 General issues regarding the cuts

There is a lot of freedom in the actual implementation of the cuts. First of all there is the choice of the different α^i and ε^i parameters. Suitable values should be found by experiments. The same is true for the question whether or not to use the logarithmic variant of the definition of δ^i. The value of the α^i parameters should clearly be nonnegative, otherwise the cut would be weaker for larger violations of the power peak. The value of ε^i could be slightly negative when α^i is positive, denoting that the cut does only cut off the current solution if the power peak violation δ^{PP} is more than $-\varepsilon^i/\delta^i$.

Another variant is the application of cuts even when the current solution is feasible. This is motivated by the fact that we may get a feasible local optimum from the optimization, where it is likely that there exist better local optima. Assuming that better local optima have a still lower power peak, or a power peak in another region of the core, cuts are applied to the node with the highest power. Since there is less justification for the validity of this cut in subsequent solutions, such a cut should only be kept for one or at most two iterations in the model.

For all types of cuts it can be advantageous to remove them after a few iterations. If they were known to cut off only bad solutions, the cuts should stay in the optimization problem forever. Due to the heuristic nature, however, it may happen that very good solutions, or the global optimal solution, are cut off. It also may happen that cuts disconnect the feasible area. Therefore, we keep them only a few iterations in the model. The precise number of iterations to keep them in the core is again to be found by experiments.

Algorithm 5.5 stops if the number of iterations in which no improved solution is found exceeds some preset number. This number is set not too large (typically 3 or 4), since we found that, once no improving solution is found in two or three iterations, the objective usually started worsening in all subsequent iterations. The algorithm terminates immediately if no cuts could be added in the current iteration, or if the solution resulting from the current iteration is exactly the same as in the previous iteration. The latter case may happen when weak cut parameters are used, so that the previous solution was not cut off from the search space.

As is shown, there is a lot of freedom in the actual implementation of the cuts. This is both an advantage and a disadvantage. It is an advantage since there is a lot of freedom in selecting the parameters so that the cut has the best effect. It is a disadvantage since it is difficult to determine which choice of parameters is the best one in a specific situation.

5.5 Test environments and solvers

We employed a number of different solvers and test environments. The programming environments used are

- The high level modeling language GAMS;
- Programs in Fortran and C using optimization software libraries;
- A Graphical User Interface (GUI), developed in Delphi Pascal and linked with some of our Fortran routines using Dynamic Link Libraries (DLL's);
- The Standard Input Format (SIF) for nonlinear optimization models.
- Web-based optimization via the NEOS server.

The first three environments are described in more detail in the forthcoming subsections, while a discussion of the latter two is postponed to Chapter 7.

We will give a brief overview of the solvers that were employed in the different environments. The most important standard nonlinear optimization packages used are

- MINOS5 (within GAMS) [37]. This solver is an implementation that uses the Reduced Gradient method combined with a Quasi-Newton method for problems with linear constraints, and a Projected Lagrangian method for the nonlinear constraints.
- CONOPT and CONOPT2 (as Fortran library and within GAMS) [27, 37]. Based on the Generalized Reduced Gradient method. Temporarily switches to a Sequential Linear Programming based method when the problem turns out to be almost linear in some part of the feasible area.
- LANCELOT (via the SIF input format) [19]. Trust region implementation that uses the Augmented Lagrangian for nonlinear constraints. Among other options, it is possible to choose between a rectangular and an ellipsoidal trust region.

Besides these solvers we have also tested several other codes but the results were not worth mentioning here. The mixed-integer optimization solvers used are

- CPLEX (linear; as C library and within GAMS) [37, 58]. It covers different variants of the simplex method as well as an interior point method for the linear part, and a Branch-and-Bound method for the integer part.

- ZOOM (linear; within GAMS) [37]. Simplex implementation combined with a Branch-and-Bound method. In general its results are inferior to that of CPLEX.
- DICOPT (nonlinear; within GAMS) [37]. This is an implementation of the Outer Approximation algorithm. For the MILP and NLP subproblems it uses the GAMS solvers that are available (MINOS5 or CONOPT for NLP, and CPLEX or ZOOM for MILP).
- The experimental MINLP code of Leyffer [73], combining an SQP algorithm and a Branch-and-Bound algorithm.

As for the linear mixed-integer subproblems, we have almost always used the solver CPLEX since it is superior to the version of ZOOM that we had available. The quality of CPLEX is influenced by the parameter settings. It turns out that the standard Branch-and-Bound strategy given by the default parameter settings can be improved for our problems by choosing alternative settings. This is discussed in more detail in the next chapter, where computational results are presented.

In nonlinear optimization the situation is less clear. Although some solvers perform consistently worse on our problems, other solvers, especially MINOS5 and CONOPT, compete with each other. In the same way as was observed for CPLEX, playing around with different solver settings influences the speed and results of the algorithms.

Rather different from all these nonlinear mixed-integer optimization packages are the global optimization packages. Some tests were performed using the global optimization package αBB [4], based on a global Branch-and-Bound method (Section 4.4.2). Two other candidate algorithms for global optimization are the BARON algorithm [112], also based on a global Branch-and-Bound method, and LGO [101], based on Lipschitzian optimization (Section 4.4.1). Limited experiments made by some of the authors of these packages indicate that only simple models, applied to small reactor cores, might possibly be solved with global optimization techniques.

5.5.1 GAMS

The greatest advantage of using a high-level modeling language is the ability to implement several modifications and adaptations of a standard model in a very short time. The user does not have to care about array definitions, sparsity structures and the like. Another important advantage is the automatic algebraic computation of first and second derivatives. Disadvantages are the longer running times, since a model has to be recompiled into a low level language readable by solver routines each time it has changed. Moreover, the communication with the underlying solvers is performed via the file system of the computer. Finally, direct computations implemented in the modeling language as needed for our fixed core evaluations are very slow.

The Outer Approximation solver DICOPT was only available within GAMS, so all tests with DICOPT were performed within GAMS. In general we implemented most of our models and model modifications the first time in GAMS. If this gave an indication that the variant might be successful, we implemented it in Fortran for further testing. We also made a Branch-and-Bound implementation in GAMS. This implementation has a few options that can be altered.

The most important one is the node selection strategy: it can perform a depth-first, breadth-first or best-bound strategy. Although the running time in GAMS cannot be used as a performance measure, the number of Branch-and-Bound nodes visited gives an indication of the efficiency of the algorithm.

5.5.2 Fortran and C codes

Most of our dedicated implementations are written in Fortran. The choice for Fortran was made since this language is adopted as a sort of standard in the Reactor Physics group of the Interfaculty Reactor Institute of the Delft University of Technology. We developed about 300kB of source code, that covers all models that are described in this thesis. For the nonlinear optimization, it invokes the NLP solver CONOPT as a subroutine library, for which our source code provides the functions and first derivatives of all objective functions and constraints.

Many implementation issues can be governed by switches in an options file. In this way, many variants can be tested and combined within the same program, without the need to recompile the source code. Regarding the main model, the following settings and model variants are supported and can be selected by setting parameters in the options file:

- ◇ the number T of time steps per cycle (Section 5.2.6);
- ◇ evenly distributed time steps or time steps that are distributed according to (5.28);
- ◇ central or forward time discretization (Section 5.2.6);
- ◇ standard model or inclusion of burnable poisons;
- ◇ standard model or the new model of Section 2.3.3;

Modules for the following solution methods and tools are available and can be invoked sequentially by the main program. The main model settings as specified in the options file are completely transparent to all these tools.

- Starting point generating routine (Section 5.1). Additional settings in the options file:
  - ◇ use starting point model 1, 2, a random starting point, or read the starting point from a file.
  - ◇ the parameter ω in starting point model 1 and 2.

  When using starting point model 1 or 2, the matrix $c_{i,\ell}$ is read from a separate file. Since these two models are optimization problems on their own, they make a separate call to CONOPT. Model 1 may be infeasible for some matrices $c_{i,l}$; in that case the program automatically invokes model 2.
- Fixed core evaluation (Section 5.1.3). Also works for fractional loading patterns. Setting in the options file:

  - ◇ the convergence tolerance ε^{pow} for the power method;

- Nonlinear continuous optimization (CONOPT). Settings in the options file:
  - ◇ the upper bound ε^{up} and objective penalty θ for the power peak relaxation variable (Section 5.2.3);
  - ◇ a relaxation parameter on the variable bounds (Section 5.2.1).

  A separate options file is used to set CONOPT-specific options. When using the new fractional model, a heuristic was implemented at the end of each iteration in CONOPT. This new fractional model converges very slowly to an optimal solution. This is partly due to the fact that a huge amount of variables must become zero in the optimal solution, and they converge very slowly to this zero value. We therefore applied a heuristic, in which those variables are manually set to zero when they are smaller than a specified tolerance. This makes the current solution slightly infeasible. Of course, this causes some extra restoration work in the next CONOPT iteration, but on average this heuristic gave a significant improvement in the convergence of the algorithm.

- Rounding heuristic (Section 5.3.1). The number of computed rounded solutions for a given fractional loading pattern is restricted to 1000.

- Pairwise interchange (Section 5.3.4). Setting in the options file:
  - ◇ Start next iteration immediately when an improved solution is found, or evaluate all neighbors and then select the best.

- Simulated annealing. This is only a draft quick-and-dirty implementation.

- Cut algorithm (Section 5.4). There are a lot of switches that can be set in the options file. For each type of cuts we may set the following options:
  - ◇ use this cut;
  - ◇ apply this cut even when a solution is feasible;
  - ◇ use the logarithmic variant like (5.35) instead of the standard variant like (5.33);
  - ◇ remove an added cut after n iterations, where n is an integer value;
  - ◇ the parameter α^c;
  - ◇ the parameter ε^c.

  Regarding the warm start after addition of cuts (Section 5.4.5), the following options can be set.
  - ◇ use model I, model II, the current point, or the original starting point, or do the complete iteration with both models I and II. In the latter case the better of the two is selected as the current solution $\bar{x}^{(1)}$ in the next iteration;
  - ◇ the maximum number P of best-known solutions that is used in model I or II;
  - ◇ linear or logarithmic objective function in model I;

- ◇ the parameter $\varepsilon^{\log}$ in model II;

Starting point models I and II are solved using a separate call to the CONOPT optimizer. The values γ_p are supplied via a separate file. A last parameter concerns the stopping criterion of the cut algorithm (Section 5.4.6):

- ◇ The maximum number of iterations without improvement of the best solution, before the algorithm terminates.

The different modules are easily combined in one main program. The parameter file automatically determines one physical model that is used in all algorithms. In this way, several algorithmic variants can be developed in a relatively easy way. A test run results in a detailed log file with the complete solution of each subalgorithm, and separate log files for the pairwise interchange algorithm and the cutting plane algorithm. Furthermore, a one line summary containing the parameter settings, the results and computation times for the different subalgorithms is added to a solution summary file, and one line containing the optimal pattern itself to a pattern summary file. In this way, we can collect the results of a number of test runs in table format in a single summary file. If a number of tests is performed with different parameter settings or different test sets, such a run is driven using a batch file (DOS) or a shell script (Unix). There is also a randomization tool that generates options files in which the parameters are randomly generated within given limits. The random seed that is used is also supplied to the main program and saved in the one-line solution summaries-file. This random seed can be fed into the randomization tool so that interesting random runs can be repeated.

Besides this Fortran system, we also have worked on a C implementation of the Branch-and-Bound algorithm. Although the basic algorithm works on a small model, we did not extend this to a fully working model for different reasons. The first and most important reason is that it turned out that our basic model almost always returned integer or almost integer solutions, so that rounding was sufficient to obtain integer solutions. Also, application of Branch-and-Bound to this model always led either to a solution that was very close to the initial solution after evaluation of very few nodes, or to very long running times with very much nodes evaluated without improvement of the best-known objective function. Very frequently, child nodes gave a better non-integer solution than their parent, until somewhere deep in the tree, all children turned out to be infeasible. This sort of oddities make the Branch-and-Bound process very difficult. Some improvement should be possible, but we decided to spent more time on improving the nonlinear optimization part. With the recently developed model of Section 2.3.3, the situation may be completely different, since now no longer the relaxed solutions are almost integer, and the model, although still nonconvex, seems to have a more smooth objective function, offering a better behavior for a Branch-and-Bound algorithm. Unfortunately, we did not have time to work this out further.

5.5.3 Graphical user interface

At some stage during the research, we felt the desire to visualize the loading patterns and the optimization process. This would lead to a better understanding of the problem. To that end, we developed a Graphical User Interface (GUI) for MS Windows, using the Delphi-Pascal language. Screen dump and descriptions of the various properties are given in Figures 5.11 to 5.14 on pages 123 to 126. The GUI is coupled to our Fortran system using Dynamic Link Libraries (DLL's), so that it always uses the latest version of our Fortran implementations for fixed core evaluation and Pairwise Interchange.

The GUI offers possibilities to change core configurations just by using drag and drop, where the effect of such changes can immediately be viewed by parameter inspection and plotting routines (see Figures 5.11 and 5.12). One may get a feeling of the shape of the objective function and the power peaking constraints in the continuous optimization by interpolating two loading patterns and evaluating the objective function in intermediate points (Figure 5.13). The Pairwise Interchange optimization routine is coupled to the system. Due to intensive interaction between the PI routine and the GUI, detailed graphical progress information is shown on the screen during a PI run (Figure 5.14). We considered the possibility of coupling the nonlinear optimization algorithm and the cut algorithm to the GUI in the same way. The major difficulty here is how to deal with non integer solutions that occur in the various stages of these algorithms. We developed a draft implementation that was able to plot fractional solutions in the main window, but this did not give clear pictures and worked very slow. Due to time limitations and other more stringent priorities we have not developed this feature further.

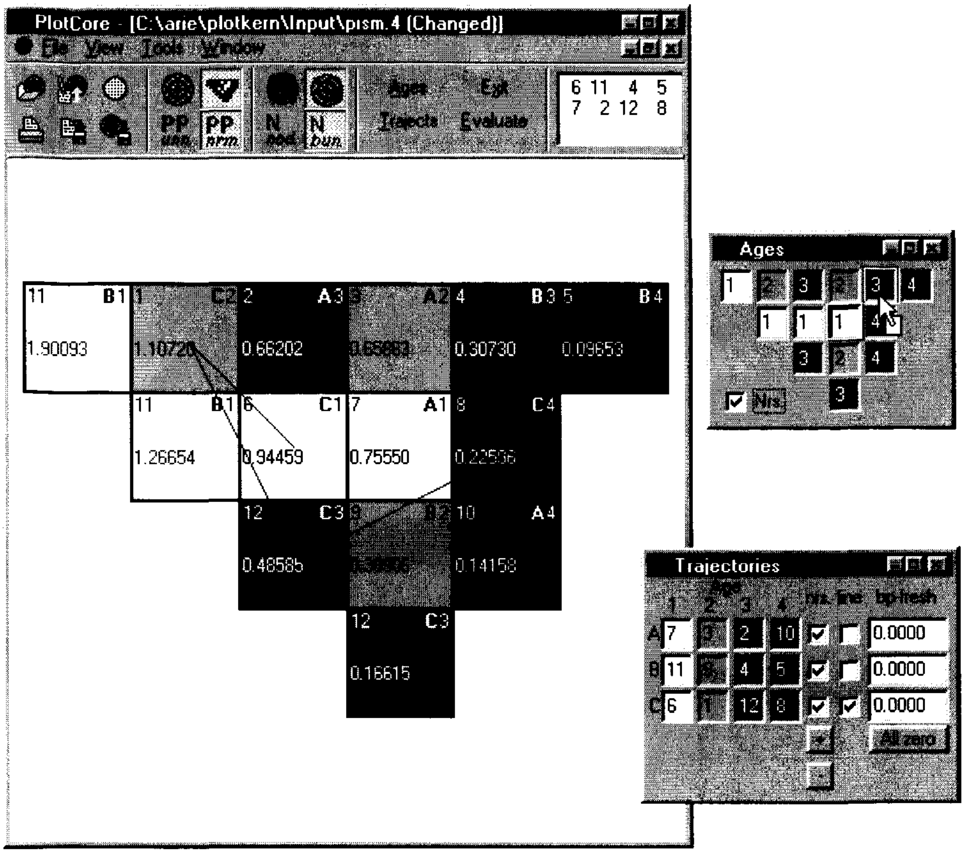
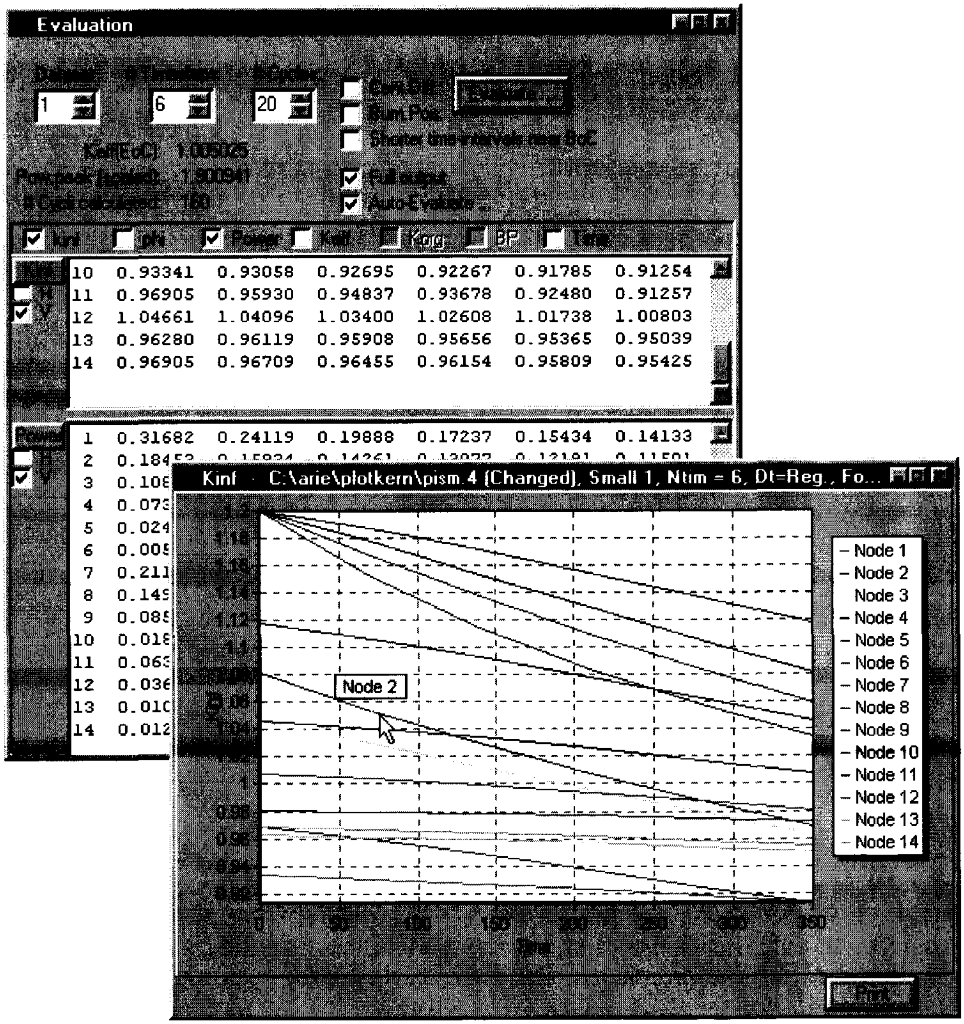


FIGURE 5.11: Main window and the two drag and drop windows of our GUI 'PlotCore'. Either the whole core or an octant core is shown in the main window. The color coding indicates the ages of the bundles. Optionally, the following numbers are shown in each node:

- The bundle number (upper left) in two possible formats, due to the octant symmetry;
- The trajectory and age (upper right), where the trajectory is indicated with a character;
- The normalized or unnormalized power peak. Power peaks exceeding the maximum allowed power peak are colored in red.

The trajectory of a bundle in the core during its lifetime can be visualized by a line, as is shown in the figure for trajectory C. In the Ages and Trajectories windows, two bundles can be interchanged by dragging one bundle to the other, as shown in the Ages window. If the appropriate check box in the evaluation window is selected (see Figure 5.12), the physical parameters are re-evaluated immediately after each drag and drop action, and the power peak values are accordingly updated. In the Trajectories window, one may specify the BP concentration in the fresh bundle of each trajectory. This concentration is optionally used in the different core analysis routines.

It is possible to input core configurations manually, or they can be read from file. The various core description formats as used by our GAMS programs and our Fortran programs are accepted, and cores can also be saved in these formats. Pictures of the cores can be printed or saved either as bitmap files, or as dedicated encapsulated postscript files.

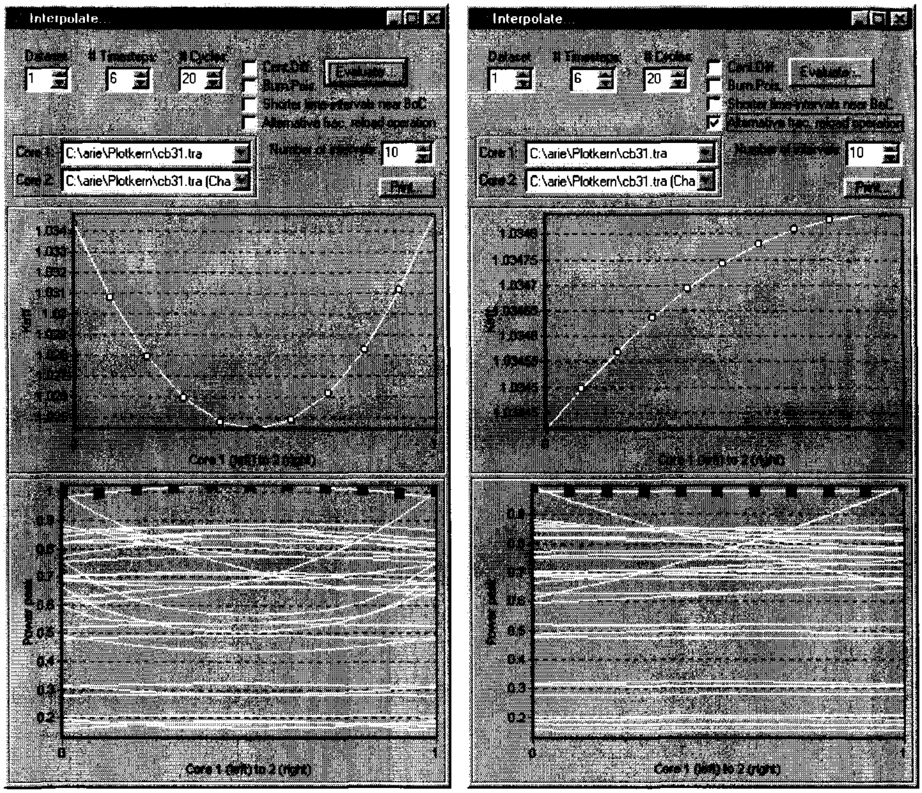


```
10  0.93341  0.93058  0.92695  0.92267  0.91785  0.91254
11  0.96905  0.95930  0.94837  0.93678  0.92480  0.91257
12  1.04661  1.04096  1.03400  1.02608  1.01738  1.00803
13  0.96280  0.96119  0.95908  0.95656  0.95365  0.95039
14  0.96905  0.96709  0.96455  0.96154  0.95809  0.95425
```

```
 1  0.31682  0.24119  0.19888  0.17237  0.15434  0.14133
 2  0.184
 3  0.10
 4  0.07
 5  0.024
 6  0.00
 7  0.211
 8  0.14
 9  0.08
10  0.01
11  0.06
12  0.03
13  0.01
14  0.01
```

FIGURE 5.12: Core evaluation window, and a plot showing the k^∞ development in time for the different nodes. One may choose between 10 available test data sets for the appropriate core geometry. Several modeling options can be set, corresponding to the switches that in our Fortran system are set via the options file. The core analysis is performed by the core evaluation routine Algorithm 5.1 from our Fortran system, that is called using MS-Windows DLL files.

After the evaluation of the core, the various physical properties can be viewed for each node and each time step. Plots can be made showing the development of these values for each node during the cycle, as is shown for the k^∞-value. In this way, one may easily visualize the effect of different modeling options, or for example the effect of different concentrations of BP.

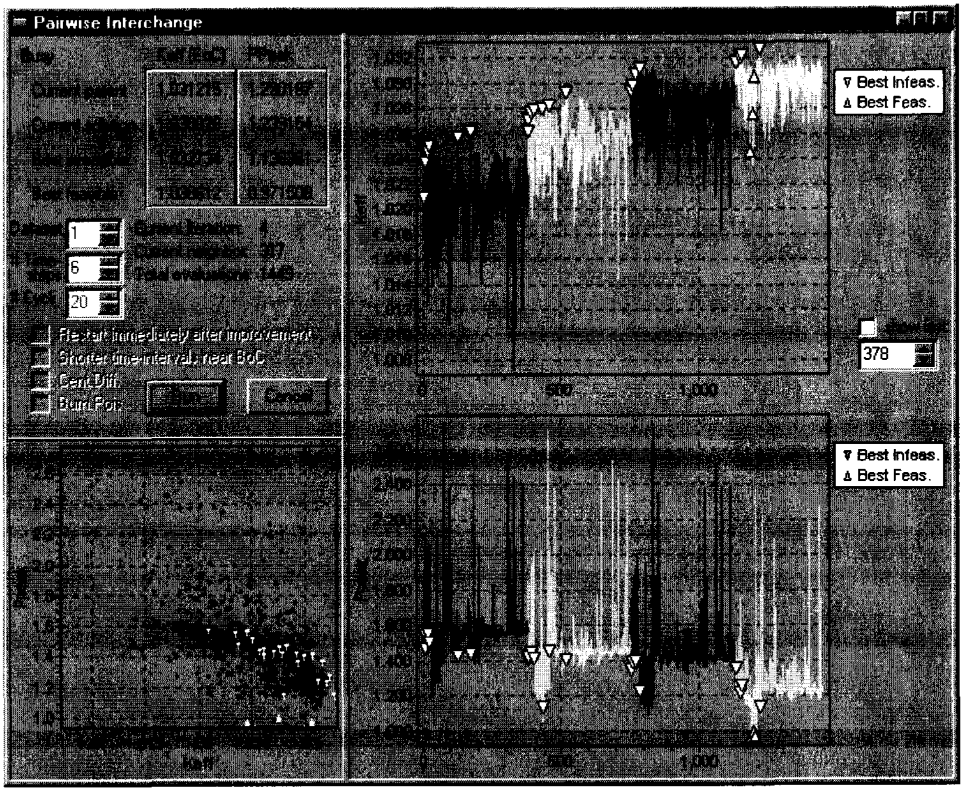


Dataset 1 · # Timesteps 6 · # Cycles 20
Core 1: C:\arie\Plotkern\cb31.tra
Core 2: C:\arie\Plotkern\cb31.tra (Cha
Number of intervals 10

Left window: ☐ CoreDiff ☐ BurnPos ☐ Shorter time-intervals near BoC ☐ Alternative bsc. reload operation

Right window: ☐ CoreDiff ☐ BurnPos ☐ Shorter time-intervals near BoC ☑ Alternative bsc. reload operation

FIGURE 5.13: Two examples of the interpolation tool in PlotCore. Given two cores with loading patterns x^1 and x^2, it evaluates a number of fractional solutions $(1-\lambda)x^1 + \lambda x^2$, where λ steps from 0 to 1. This gives some feeling of the shape of the objective function and the power peaking constraints in the search space for the continuous nonlinear optimization model. In the left example, where the basis model is used, the k^{eff}-plot suggests that the two integer patterns are local optimal solutions. In fact, such an objective shape is obtained for many pairs of patterns, which suggests that there are many patterns that are local optima. The example in the right window shows the same example solved with the new fractional model as developed in Section 2.3.3. At $\lambda = 0$ and $\lambda = 1$ all values are the same as for the basic model, but otherwise the figures are very different. The integer patterns are no longer local optima. Also note that all power peaking values are less than 1 for all λ, while in the basic model, all fractional patterns with $\lambda \in [\approx 0.2; \approx 0.8]$ have too high power peak. For these reasons, better results are to be expected with this model.



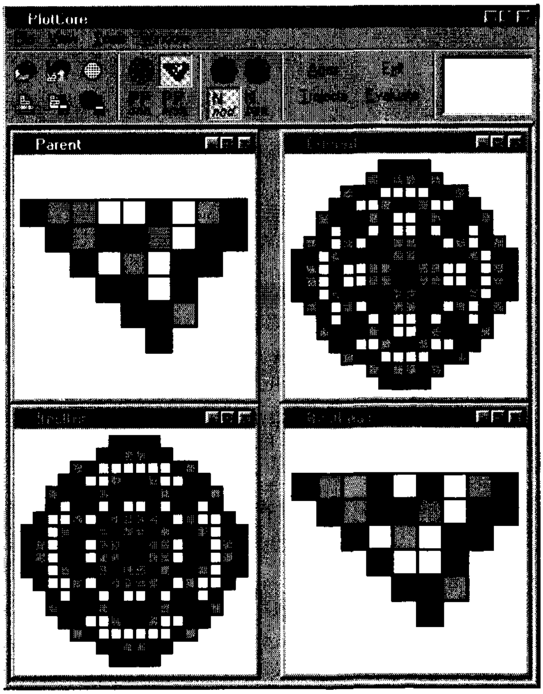
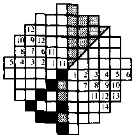


FIGURE 5.14: Pairwise interchange control window and the corresponding four cores in the main window. These cores are updated during the iterations. When a new parent is selected and a new main iteration is started, the color in the two iteration plots is changed. In the example, the first feasible solution was found during the fourth main iteration, immediately followed by two subsequent improved feasible patterns, indicated with upright triangles in the iteration plots. It is interesting to see that both the average k_{EoC}^{eff} and the average maximal power peak are improving each time a new neighborhood is explored. Note also that this improvement for k_{EoC}^{eff} becomes less pronounced at each subsequent change of neighborhood, since PI starts with selecting that exchanges as new parents that give the most improvement in objective value.



Chapter 6

Computational results

This chapter contains results on the different models, solvers and algorithms that were discussed in the previous chapters. We used a set of 40 different test problems that are described in Section 6.1. Results for different modeling variants and solvers for the basic model are presented in Section 6.2. The results for the model with Burnable Poisons are discussed in Section 6.3. The chapter ends with some summarizing and concluding remarks in Section 6.4.

Table 6.1 presents the different computer systems on which the different tests are performed, and the different versions of compilers and software that were used. Unless stated explicitly, the results presented in this chapter are obtained on the HP workstation.

	Intel Pentium II PC	HP9000C200	Cray J90SE
Processor	P-II 233 MHz	PA-8200 200 MHz	10 CPU's at 200MHz
Memory	64 MB	512 MB	2048 MB
Op. System	MS Windows 95/98*	HP-UX B.10.20	Unicos 10.0.0.1
Compilers	Lahey Fortran LF90 4.0 Borland Delphi 4	HP-UX Fortran 77	Cray CF90 Fortran
Software	GAMS 2.25.087 using	GAMS 2.50.093 using	
	- DICOPT++ 4-4	- DICOPT++ HP-UX/9000-8-C	
	- CONOPT 2.040-017	- CONOPT 2.070A	
	- MINOS 5.3	- MINOS 5.4	
	- ZOOM-XMP 2.2	- ZOOM-XMP 2.2	
		- CPLEX 6.0	
	CONOPT 2.069A library	CONOPT 2.066F library	
		CPLEX 6.0 library	CPLEX 4.0 library

* Computation time statistics for Windows 95/98 are not accurate, since it is not possible to measure the CPU time for each process separately.

TABLE 6.1: The various computing environments.

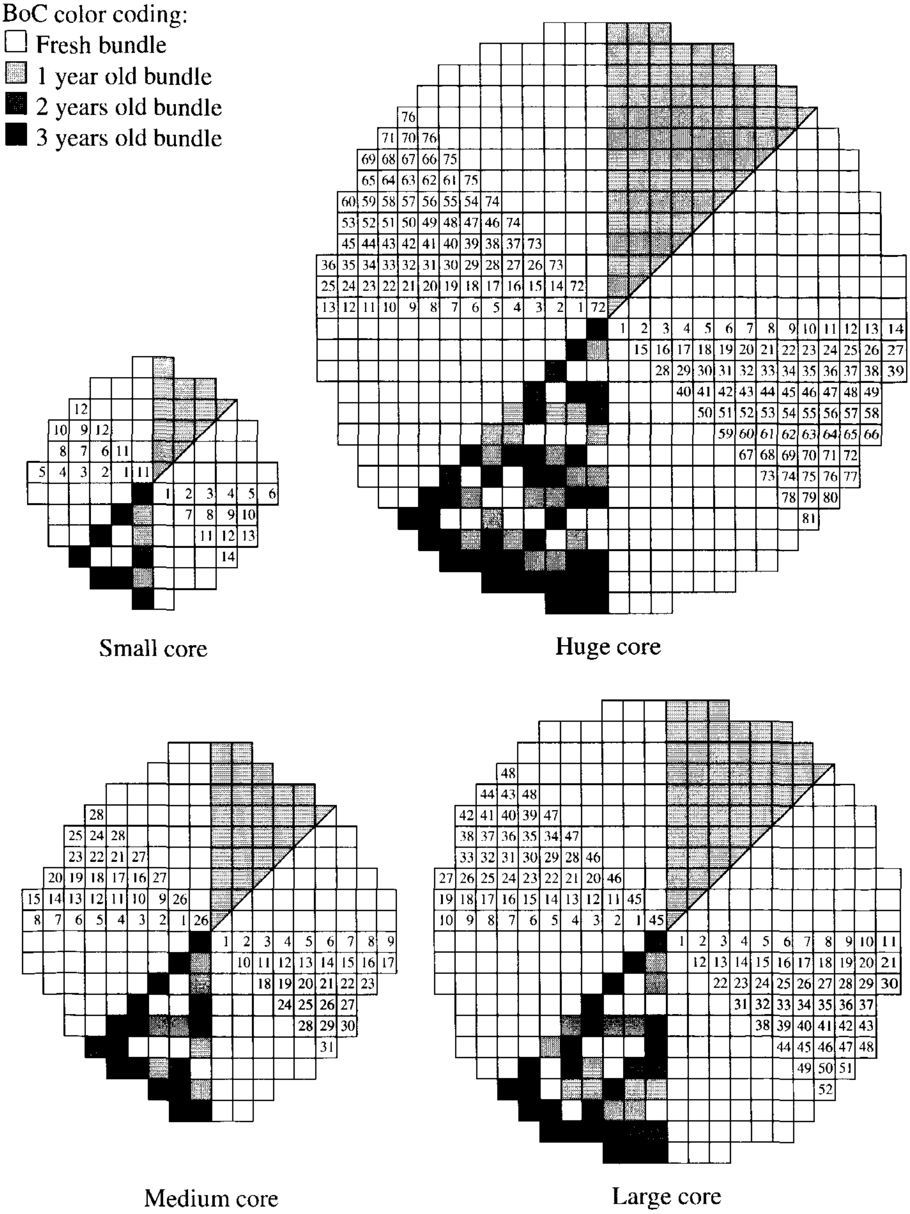
6.1 The test sets

We used four different test core layouts, differing in the number of fuel positions in the core. They are shown in Figure 6.1 on page 129. The core geometries are designed in such a way that the number of fuel positions in an octant core is a multiple of four. In this way we can restrict ourselves to equilibrium cycles where the four different age groups have equal size. For each of the four core geometries, we have defined ten data sets, defining interaction probabilities between fuel bundles, and parameters like the required power level, the cycle time and the power peaking factor. Using more data sets is a good practice in testing algorithms, since dependence of the results on peculiarities of one data set is reduced in this way.

The values of the parameters for the different test sets are given in Table 6.2. The values in the node interaction matrix $G_{i,j}$ depend on the fast diffusion length L_f (see (2.10)), and the geometrical dimensions of the nodes. Our method of computing the matrix G is described in [40]. It is based on the approximation that the value $G_{i,j}$ is equal for any pair i, j with the same relative distance in the core. The probabilities $G_{i,j}$ for a fixed node j are described with the values g^0, g^{11}, g^{12}, g^{21}, g^{22} and g^{23}, as shown in Figure 6.2. From these 6 parameters, the whole matrix G can be computed. The values of $g^0, \cdots, g^{23}$ for all test sets are given in Table 6.3.

The 'Huge' test sets are to our knowledge larger then any existing PWR reactor core. The computation times for the different algorithms for the Huge problems are very large. For these reasons, the Huge test sets are only used in a few of the computational tests that are presented in this chapter.

Table 6.4 shows the dimensions of the different MINLP models that are solved in this chapter for all the test sets. These dimensions are based on the use of an octant core, where the diagonal nodes are pairwise filled with the same bundle. The number of binary variables is not stated explicitly in the table but is equal to $12^2 = 144$ for the Small models, $28^2 = 784$ for the Medium models, $48^2 = 2304$ for the Large models, and $76^2 = 5776$ for the Huge models.



Small core

Huge core

Medium core

Large core

FIGURE 6.1: The four different core layouts. The total volume of an octant core corresponds to a multiple of four bundles. Diagonal positions are shared by two octants. To reduce the error due to octant symmetry, we impose that two adjacent diagonal nodes must share the same bundle, *cf.* the numbering in the upper left corner of each layout. The lower right corners show the numbering of the individual nodes. The lower left corner shows an example bundle assignment, where the gray values represent the ages of the bundles.

Test set	α	k^{fresh}	P_c	cyc. time	α^p	f^{lim}
small 1	6.0E-6	1.2	364.0	350.0	7.5E-4	2.0
3	7.0E-6	1.26	380.0	350.0	7.5E-4	1.8
2,4,5	6.0E-6	1.25	364.0	350.0	7.5E-4	1.8
6-10	6.0E-6	1.25	380.0	350.0	7.5E-4	1.8
medium all	6.0E-6	1.3	1200.0	350.0	7.5E-4	1.75
large all	6.0E-6	1.2	1200.0	350.0	7.5E-4	1.9
huge all	6.0E-6	1.2	2000.0	350.0	7.5E-4	1.8

TABLE 6.2: Parameter values for the different test sets. The values α and α^p in this table are not exactly the values from (2.34) and (2.46), but they have still to be multiplied by P_c.

.003	.007	.008	.007	$g^{23}=$ .003
.007	.025	.050	$g^{12}=$ .025	$g^{22}=$ .007
.008	.050	$g^{0}=$ .600	$g^{11}=$ .050	$g^{21}=$ .008
.007	.025	.050	.025	.007
.003	.007	.008	.007	.003

FIGURE 6.2: Probabilities $G_{i,j}$ that neutrons produced in center node j are absorbed in surrounding nodes i. Matrix G is determined by the values $g^0, g^{11}, g^{12}, g^{21}, g^{22}$ and g^{23}.

Test	Small test sets						Medium, Large and Huge test sets					
set	g^0	g^{11}	g^{12}	g^{21}	g^{22}	g^{23}	g^0	g^{11}	g^{12}	g^{21}	g^{22}	g^{23}
nr 1	0.684	0.0650	0.0140	0	0	0	0.520	0.0620	0.0330	0.0080	0.0070	0.0030
2	0.500	0.0800	0.0450	0	0	0	0.520	0.0600	0.0350	0.0080	0.0070	0.0030
3	0.500	0.0800	0.0450	0	0	0	0.500	0.0650	0.0350	0.0080	0.0070	0.0030
4	0.600	0.0500	0.0250	0.0080	0.0070	0.0030	0.480	0.0670	0.0380	0.0080	0.0070	0.0030
5	0.560	0.0560	0.0290	0.0080	0.0070	0.0030	0.460	0.0700	0.0400	0.0080	0.0070	0.0030
6	0.580	0.0530	0.0270	0.0080	0.0070	0.0030	0.520	0.0600	0.0350	0.0120	0.0050	0.0030
7	0.510	0.0640	0.0310	0.0080	0.0070	0.0055	0.510	0.0640	0.0310	0.0080	0.0070	0.0055
8	0.500	0.0670	0.0340	0.0080	0.0070	0.0020	0.500	0.0670	0.0340	0.0080	0.0070	0.0020
9	0.470	0.0700	0.0375	0.0080	0.0070	0.0030	0.470	0.0700	0.0375	0.0080	0.0070	0.0030
10	0.440	0.0750	0.0400	0.0080	0.0070	0.0030	0.440	0.0750	0.0400	0.0080	0.0070	0.0030

TABLE 6.3: Elements of the G-matrix for the 10 data sets with different core geometries. Note that the same g-values lead to different G-matrices in the different core geometries.

	6 time steps						10 time steps						20 time steps					
	#VARS	#CONS	#LNZ		#NLNZ		#VARS	#CONS	#LNZ		#NLNZ		#VARS	#CONS	#LNZ		#NLNZ	
			Forw	Cent	Forw	Cent			Forw	Cent	Forw	Cent			Forw	Cent	Forw	Cent
Small																		
Basic Model	318	624	372	302	3270	3410	434	848	428	302	4302	4554	724	1408	568	302	6882	7414
prob. 4-10					4086	4226					5662	5914					9602	10134
Keoc Model	327	633	423	353	[illegible]	2146	443	857	479	353	3038	3290	733	1417	619	353	5618	6150
prob. 4-10					[illegible]	2962					4398	4650					8338	8870
BP Model	405	792	540	400	3662	3942	577	1128	708	456	4918	5422	1007	1968	1128	596	8058	9122
prob. 4-10					4478	4758					6278	6782					10778	11842
Frac. Model	1326	1632	2430	1590	[illegible]	6630	2114	2528	3830	2318	7214	10238	4084	4768	7330	4138	12874	19258
prob. 4-10					[illegible]	7446					8574	11598					15594	21978
+ Eps	1	0	84	84			1	0	140	140			1	0	280	280		
+ Keff constr	0	5	10	10			0	9	18	18			0	19	38	38		
Medium																		
Basic Model	1162	2312	1754	1599	26010	26320	1414	2808	1878	1599	30454	31012	2044	4048	2188	1599	41564	42742
Keoc Model	1183	2333	1992	1837	[illegible]	9455	1435	2829	2116	1837	13589	14147	2065	4069	2426	1837	24699	25877
BP Model	1355	2684	2126	1816	27126	27746	1731	3428	2498	1940	32066	33182	2671	5288	3428	2250	44416	46772
Frac. Model	6370	7520	12387	8047	[illegible]	44486	10094	11488	19455	11643	46946	62570	19404	21408	37125	20633	74796	107780
+ Eps	1	0	186	186			1	0	310	310			1	0	620	620		
+ Keff constr	0	5	10	10			0	9	18	18			0	19	38	38		
Large																		
Basic Model	2934	5856	4920	4660	105728	106248	3354	6688	5128	4660	114160	115096	4404	8768	5648	4660	135240	137216
Keoc Model	2970	5892	5580	5320	[illegible]	20408	3390	6724	5788	5320	28320	29256	4440	8804	6308	5320	49400	51376
BP Model	3258	6480	5544	5024	108120	109160	3886	7728	6168	5232	117384	119256	5456	10848	7728	5752	140544	144496
Frac. Model	17910	20832	35496	23016	[illegible]	161160	28314	31648	55672	33208	164184	209112	54324	58688	106112	58688	234144	328992
+ Eps	1	0	312	312			1	0	520	520			1	0	1040	1040		
+ Keff constr	0	5	10	10			0	9	18	18			0	19	38	38		
Huge																		
Basic Model	6754	13496	12038	11633	379980	380790	7406	14792	12362	11633	394080	395538	9036	18032	13172	11633	429330	432408
Keoc Model	6811	13553	13634	13229	[illegible]	39981	7463	14849	13958	13229	53271	54729	9093	18089	14768	13229	88521	91599
BP Model	7259	14468	13010	12200	384840	386460	8235	16412	13982	12524	400236	403152	10675	21272	16412	13334	438726	444882
Frac. Model	43690	50432	87449	56669	[illegible]	521892	68966	76352	137021	81617	523032	633840	132156	141152	260951	143987	679782	913710
+ Eps	1	0	486	486			1	0	810	810			1	0	1620	1620		
+ Keff constr	0	5	10	10			0	9	18	18			0	19	38	38		

TABLE 6.4: Number of variables, constraints and nonzeros for the different models and test sets, for 6, 10 and 20 time steps, both when using forward and central discretization. The small test sets 4-10 have more nonzeros than set 1-3 since there are more nonzeros in the interaction matrix G, *cf.* Table 6.3. The terms Keoc model, +Eps and +Keff constr refer to the reformulations on page 134. Frac. model refers to the model of page 138. BP model *refers to the* extension with burnable poisons (Section 6.3).

6.2 Results for the basic models

In this section the results obtained with different models, different algorithms and different solvers are presented and discussed. We start with results that are used to compare different modeling variants. Based on these results, we compare the two most competitive solvers for the nonlinear part of the problem in Section 6.2.2. In Section 6.2.3 the different settings of different mixed-integer solution strategies are compared, and the best settings of the different strategies are used for testing all the different algorithms thus far. The effects of the addition of cuts in order to find improved solutions are discussed in Section 6.2.4.

6.2.1 Different modeling features and modeling variants

This section shows how the nonlinear optimization problem is effected by different extensions and adaptations to the basic model. Most results in this section are obtained with the nonlinear optimization package CONOPT, combined with our rounding procedure. In each test a certain feature is evaluated, while all other settings are kept the same. In the text, we give a short description of what is compared, together with a discussion of the results. The actual results, together with an additional description of each test, are presented in the tables and figures.

Starting point

In Section 5.1 we developed an algorithm for computing the starting point that depends on a location matrix with binary components $c_{i,\ell}$, specifying whether location i is regarded as a suitable candidate place for a bundle of age ℓ. Also a parameter $\omega \in [0,1]$ has to be provided specifying how close the starting point should follow the specification of $c_{i,\ell}$. Results for different starting points $c_{i,\ell}$ and different values of ω are presented in Figure 6.3. From the results it can be concluded that it is important to have an initial solution that possesses a ring structure. It is also important to have a suitable large value for ω, so that the starting point indeed resembles the ring structure that was proposed by the $c_{i,\ell}$ matrix.

A striking result is that even for the bad $c_{i,\ell}$-matrix 5, the larger values of ω lead to better results. In the forthcoming results in this chapter, we use always $c_{i,\ell}$-matrix number 0 with ω-values in the range $[0.8, 1.0]$.

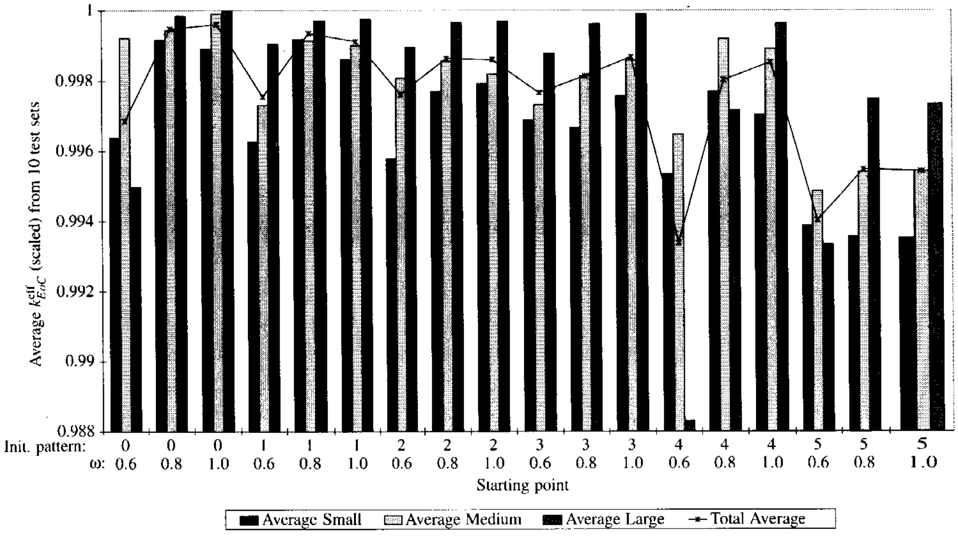


FIGURE 6.3: Comparison of different starting points. The numbers 0 to 5 in the top row of the horizontal axis labels correspond to different $c_{i,\ell}$-matrices, *cf.* Section 5.1. Matrices 0 to 4 have more or less a ring structure, matrix 5 is random. The starting point is computed using starting point model I, or if this is infeasible, starting point model II is used. For each test set, the best result from all starting points is set to 1, and the other results are scaled to this best solution. Infeasible solutions are given a k_{EoC}^{eff} of 1. A bar in the graph is the average scaled result for the 10 test sets of the corresponding problem size.

Reformulations

In Table 6.5 on page 135, the effect of the following three reformulations is shown:

- Power peaking constraint relaxation *cf.* Section 5.2.3, with parameters $\varepsilon^{\text{up}} = 1.0$ and $\theta = 1.0$;
- Addition of the linear constraints $k_{t+1}^{\text{eff}} \leq k_t^{\text{eff}}$, $t = 1, \cdots, T-1$ *cf.* Section 5.2.2.
- Addition of extra variables k^{EoC} to eliminate trilinear terms, *cf.* Section 5.2.5.

The differences in results obtained from the different reformulations in Table 6.5 are quite small. An indication about the differences is obtained by defining an average quality of the reformulations as follows. For each test problem, the best objective value obtained for a test problem was scaled to 1. The other results for this test problem were scaled according to this objective value, resulting in a value slightly smaller than one (infeasibility is given a value 0). The average of these scaled objective values per model is listed at the bottom of the table, together with the number of problems in which the model gave the best objective value. For the computation times a similar normalization was done. Here a reformulation is considered better if the average normalized computation time is smaller.

No reformulation is always better than the others. The formulations without power peak relaxation lead however too often to infeasibility, so relaxing the power peaking constraints is necessary.

The effect of adding the linear constraints on k^{eff} can be seen by comparing the second an fourth model. The model 'Both' gives an almost negligible improvement in the average quality, and a somewhat larger improvement of the computation time. The differences are so small, however, that one hardly can conclude that these additional constraints are useful.

The k^{EoC} model leads on average to a slightly worse objective function and a slightly better computation time. We found that this model reformulation had different effect for different solvers, as is shown in Figure 6.4 on page 136. For MINOS5, the reduction of computation time for the k^{EoC} model is much more pronounced, especially for the larger problems. This can be explained by the reduced number of nonzero's in the k^{EoC} model. The computation time reduction becomes more pronounced for larger problems than for small problems, which is consistent with the reduction in the number of nonzeros, *cf.* Table 6.4. The results indicate that CONOPT is more stable and faster on both models. Since for CONOPT the gain in computation time is much less pronounced, while the quality slightly decreases (see Table 6.5), none of the models is a clear winner. In further tests in this chapter, we used the model without k^{EoC} variables.

	k^{eff}_{EoC}					time (s)				
nr.	Standard	Ppeak ε	k^{eff}	Both	k^{EoC}	Standard	Ppeak ε	k^{eff}	Both	k^{EoC}
small 1	**1.00983**$^+$	1.00883	**1.00983**$^+$	1.00832	1.00898$^+$	**1.0**	11.0	**1.0**	4.8	**1.0**
2	1.04711*	1.04320	1.04711*	1.04330	**1.04860**	5.6	3.9	5.7	7.7	**3.3**
3	**1.02033**$^+$	1.01917$^+$	**1.02033**$^+$	1.01976$^+$	1.01807$^+$	1.1	1.1	1.1	1.4	**1.0**
4	1.03929*	1.04269$^+$	1.03929*	**1.04273**$^+$	1.04035	3.7	**1.0**	3.7	1.3	4.8
5	1.03963$^+$	**1.04045**	1.03963$^+$	**1.04045**	1.03952$^+$	1.6	3.8	1.6	4.2	**0.9**
6	1.03317$^+$	**1.03402**$^+$	1.03317$^+$	**1.03402**$^+$	1.03371$^+$	1.4	**1.1**	1.4	**1.1**	1.2
7	1.02682	1.02203	1.02682	**1.02739**	1.02566	**7.6**	42.5	**7.6**	102.5	11.9
8	1.02938*	1.02931	1.02938*	**1.02947**	1.02771	6.6	**4.6**	6.6	9.1	5.1
9	1.02571$^+$	**1.02720**	1.02571$^+$	1.02198	1.02588	**1.6**	21.4	**1.6**	4.6	4.6
10	1.02415	**1.02573**	1.02415	1.02519	1.01817	7.6	5.4	7.6	47.3	**4.9**
medium 1	1.03432$^+$	1.03489	1.03432$^+$	**1.03514**	1.03503	**22.2**	133.8	22.3	50.2	91.8
2	1.03427*	1.03442	1.03427*	1.03441	**1.03477**	83.1	1858.1	82.7	45.1	**37.8**
3	1.03439*	1.03470	1.03439*	1.03470	**1.03480**	53.6	48.4	53.7	45.3	**34.7**
4	1.03342*	1.03385$^+$	1.03342*	1.03385$^+$	**1.03390**	52.8	**15.3**	52.6	15.4	39.1
5	1.03293*	**1.03362**$^+$	1.03293*	**1.03362**$^+$	1.03339$^+$	116.9	20.7	117.2	20.7	**14.3**
6	1.03522*	1.03526	1.03522*	**1.03540**	1.03485	49.3	144.7	**49.1**	69.5	873.6
7	1.03311*	1.03357	1.03311*	1.03357	**1.03394**	82.9	44.3	82.4	**42.0**	194.9
8	1.03519$^+$	**1.03534**	1.03519$^+$	1.03533	1.03503	25.9	46.7	**25.7**	48.0	61.2
9	1.03337*	1.03379	1.03337*	1.03379	**1.03384**	143.0	56.9	143.2	**54.3**	1592.0
10	1.03240*	**1.03317**	1.03240*	**1.03317**	1.03313$^+$	67.0	54.7	66.9	54.0	**14.5**
large 1	**1.04741**$^+$	1.04730	**1.04741**$^+$	1.04729	1.04720	**155.8**	367.9	159.7	372.9	174.7
2	1.04728	**1.04735**	1.04728	1.04731	1.04734$^+$	271.0	357.7	271.8	339.2	**91.0**
3	**1.04707**	1.04702	**1.04707**	1.04704	**1.04707**	563.6	293.9	565.3	579.7	**177.2**
4	**1.04667**	1.04658	**1.04667**	1.04658$^+$	1.04610$^+$	421.6	338.0	422.2	159.6	**105.1**
5	1.04633	**1.04639**	1.04633	1.04638	1.04626$^+$	353.9	1300.1	353.2	532.7	**108.4**
6	1.04738	**1.04742**	1.04738	1.04735	1.04723$^+$	270.9	352.5	270.0	1070.3	**98.4**
7	1.04654	1.04664	1.04654	**1.04665**	1.04659	314.5	343.8	314.2	384.9	**160.0**
8	**1.04728**	1.04707	**1.04728**	1.04707	**1.04728**	243.7	351.0	243.3	296.6	**222.2**
9	1.04645$^+$	**1.04663**	1.04645$^+$	1.04661$^+$	1.04644	153.5	627.4	**153.0**	159.5	161.5
10	1.04609	**1.04617**$^+$	1.04609	**1.04617**$^+$	1.04603$^+$	519.9	162.0	525.5	165.7	**86.0**
av. quality	0.63311	0.99952	0.63311	0.99953	0.99929	2.348	4.890	2.351	3.104	3.048
# times best	6	12	6	11	8	5	4	6	3	17

*=Feasible relaxed optimum found, but no feasible integer optimum found.

$^+$=Relaxed optimum is integer.

Results are with DICOPT, using CONOPT as NLP solver and CPLEX with depth first branching as MILP solver.

TABLE 6.5: The effect of different model variants:

1. Standard: The basic model without modifications (except variable bounds).
2. Ppeak ε: The basic model where the power peak constraint is relaxed with ε *cf.* Section 5.2.3, with parameters $\varepsilon^{\text{up}} = 1.0$ and $\theta = 1.0$.
3. k^{eff}: The basic model with addition of the linear constraints $k^{\text{eff}}_{t+1} \leq k^{\text{eff}}_{t}$, $t = 1, \cdots, T-1$ as explained in Section 5.2.2.
4. Both: A combination of 2 and 3.
5. k^{EoC}: The model of 4 with addition of extra variables k^{EoC} to replace the trilinear terms by bilinear terms, *cf.* Section 5.2.5.

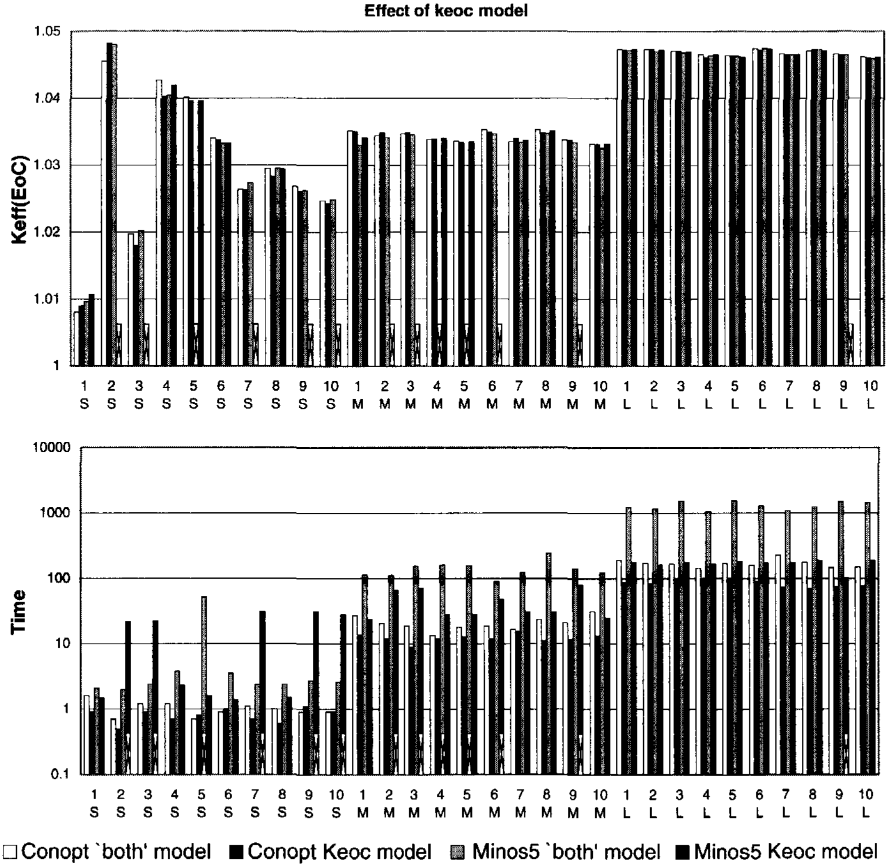


FIGURE 6.4: Comparison of the Basic model and the k^{EoC}-model. Both are solved using CONOPT and MINOS5. Results are for the continuous optimization only; the solution may be non-integer. Crosses in the figure indicate that no solution was found within the iteration limit, due to convergence problems.

Time discretization

As we have seen in Section 5.2.6, a finer time discretization, or the application of central discretization instead of forward discretization will lead to more accurate objective values for fixed core evaluation. This does not imply that a better optimal solution will be found. In Figure 6.5 on page 137 we show the results of CONOPT followed by rounding with four different types of discretization. Although there are some cases in which central discretization gives better solutions, there are also cases where the opposite is true. For the further tests, we restricted ourselves to the model with 6 time points and forward discretization. The effect on the solution found is not dramatic, and the total testing time is reduced.

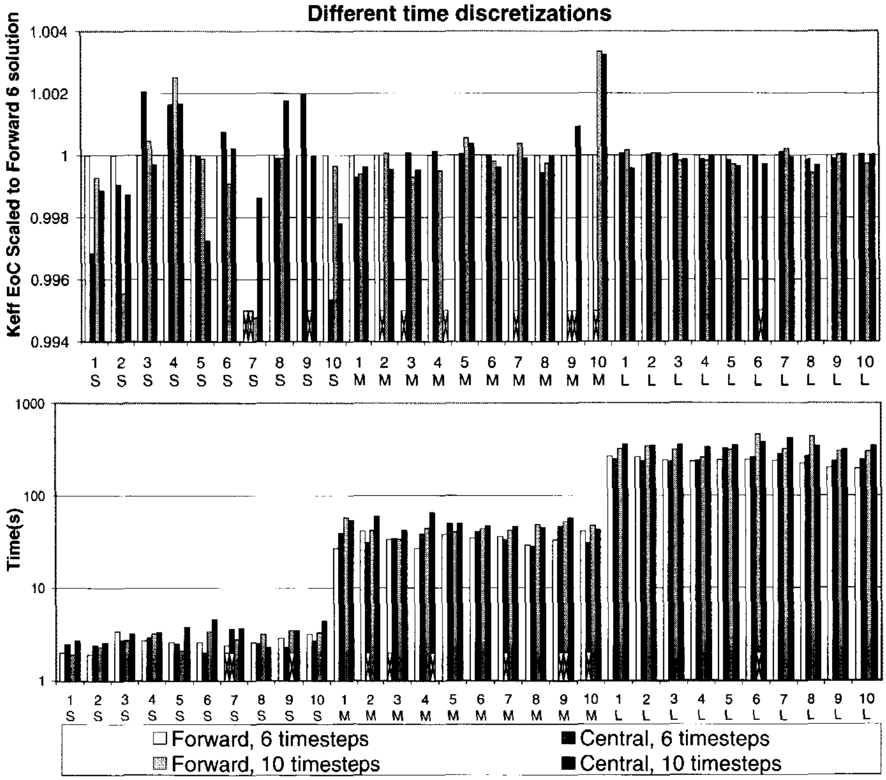


FIGURE 6.5: Conopt+Rounding algorithm with forward and central discretization using 6 and 10 time steps per cycle, a total of four different cases. Solutions for the basis model for 30 test sets. Since different time discretizations lead to different values of k_{EoC}^{eff} even for the same pattern, the value k_{EoC}^{eff} at the optimal pattern is recomputed using central time discretization and 10 time steps in all cases. The objective values are scaled to the solution of forward time discretization with 6 time steps for each test set.

New fractional model

As is suggested several times in the previous chapters, the new model developed in Section 2.3.3 relieves some difficulties in nonlinear optimization, since the shape of the objective function becomes much more smooth. This is illustrated by Figure 5.13 on page 125, and by the cover illustration of this thesis, which should be read as follows. The bottom plane is a two-dimensional representation of (parts of) the search space. The circles are integral loading patterns. A triangle between three circles contains all convex combinations of these three loading patterns. The lower (dark) function is the objective function of the basic model, while the upper (gray) one is the objective function of the new fractional model. It can be seen that, within this subset of the search space, each integral loading pattern is a local optimal solution in the basic model, hence the result is very dependent on the starting point. The objective function of the new fractional model is much more smooth, and even when starting in some bad integral solution, the optimization may go away to the neighborhood of some other integral solution, or to a fractional local optimal solution. Further, this picture demonstrates why the old model tends to produce solutions with only a small number of non-integer variables while the new model tends to give almost entirely fractional solution.

Although, due to time limitations, we did not have much opportunity for testing with this model, results for the basic model with CONOPT + rounding are given in Table 6.6. As could be expected, the number of non-binary values in the continuous optimum of the new model is much larger than in the basic model. The rounded optimal loading patterns obtained with the new model are almost always better than those obtained with the basic model. The price is a significant increase in the computation time, that frequently supersedes the computation time of the Pairwise Interchange algorithm. The largest part of this computation time is spent in the first NLP, which is caused by the increased number of variables and constraints, and slow convergence near the optimal points. This slow convergence is illustrated by Figure 6.6, where for one test problem the objective function as function of the CONOPT iteration is shown. The little peaks towards the end are slightly infeasible points, indicating that CONOPT has difficulties in convergence. A possible way to work around this convergence problem is based on the observation that, when the curve in Figure 6.6 starts flattening, many of the binary variables that are zero in the final solution have already a very small value, while many of the binary variables that should become one are already close to one. A rounding routine may be applied that interrupts the NLP procedure as soon as the curve is 'sufficiently flat' during a number of iterations. More research is needed to determine when and how this rounding routine should interact with the NLP solver.

Several results in Table 6.6 are marked with $^{@}$, indicating that rounding gave no integer solution. This problem is not severe. Typically, the power peaking constraint is slightly exceeded in the rounded solution, and a quick test showed that performing one PI iteration leads usually very rapidly to a feasible solution. An alternative way to get a feasible solution is the addition of Burnable Poison (Section 6.3).

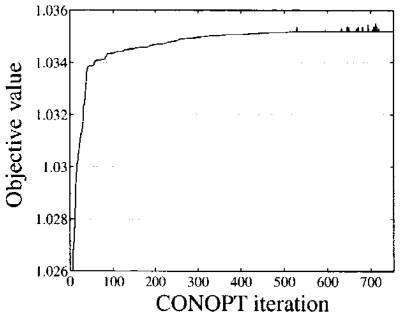


FIGURE 6.6: Objective value as function of the iteration for the fractional model and test set Medium 3.

	NLP optimum		# Noninteger x		# Rounded sol.s		Rounding optimum		Total time	
nr.	Basic	Frac	Basic	Frac	Basic	Frac	Basic	Frac	Basic	Frac
Small 1	1.01072^{+}	**1.01167**	0	19	1	18	1.01072	**1.01082**	**1.7**	21.3
2	1.04799^{+}	**1.04853**	0	14	1	9	1.04799	**1.04839**	**1.8**	10.6
3	1.01681	**1.02069**	4	0	2	1	1.01500	**1.02069**	**2.5**	9.6
4	1.03882	**1.04230**	10	20	4	16	1.04024	**1.04197**	**2.5**	27.9
5	1.03959^{+}	**1.04039**	0	16	1	2	**1.03959**	1.03712	**2.0**	8.5
6	1.03065	**1.03407**	8	4	4	2	1.03312	**1.03407**	**2.9**	14.8
7	1.02636	**1.02793**	8	14	4	9	**1.02222**	$1.02785^{@}$	**2.3**	18.1
8	**1.02885**	1.02280^{*}	4	97	2	1000	1.02764	**1.02800**	**2.4**	74.1
9	1.02601	**1.02751**	8	10	4	4	$1.02759^{@}$	$1.02749^{@}$	**3.5**	16.9
10	1.02484	**1.02552**	7	15	3	3	1.02490	**1.02511**	**2.2**	15.4
Medium 1	1.03383	**1.03587**	15	54	13	1000	1.03360	**1.03460**	**29.7**	1831.7
2	1.03474	**1.03580**	0	41	1	289	**1.03474**	$1.03526^{@}$	**34.5**	1735.6
3	1.03222	**1.03526**	8	53	4	605	$1.03204^{@}$	**1.03411**	**47.1**	1496.6
4	1.03396	**1.03471**	4	47	2	492	**1.03396**	$1.03419^{@}$	**35.3**	2084.8
5	1.03266	**1.03417**	4	44	2	345	1.03264	**1.03303**	**32.4**	1624.8
6	1.03490	**1.03612**	4	46	2	401	$1.03494^{@}$	**1.03452**	**44.1**	1713.7
7	1.03307	**1.03463**	4	49	2	310	$1.03384^{@}$	$1.03409^{@}$	**43.9**	2167.9
8	1.03436	**1.03581**	4	52	2	378	**1.03446**	$1.03519^{@}$	**25.5**	1455.9
9	1.03298	**1.03456**	4	39	2	165	$1.03370^{@}$	**1.03401**	**33.6**	1283.5
10	1.03244	**1.03372**	7	40	3	264	**1.03207**	$1.03341^{@}$	**38.5**	1599.3

*=Relaxed solution is infeasible.
$^{+}$=Relaxed optimum is integer.
$^{@}$=No feasible rounding found.

TABLE 6.6: Results of the new fractional model with Rounding algorithm. The column 'Basic' contains the results for the basic model, the column 'Frac' shows the results for the new fractional algorithm. The first columns give the objective value of the optimal solution of the continuous NLP. The column '# non-integer x' gives the number of x-variables in the solution that was non-integer. The next column shows the number of roundings of the continuous optimum that were evaluated, and the rounding optimum is the objective value of the best rounded pattern found. The maximum number of rounded solutions that was evaluated was limited to 1000.

Summary

It can be concluded from the different results obtained in this section that addition of problem-specific features to the model is necessary to obtain good-quality solutions. In our case, this denotes that ring-structured starting patterns and power peak relaxation were necessary to obtain stable results. Other extensions such as the addition of the k^{EoC} variables do not harm the solution process, but are not of much use either. This last example further shows that the benefit from a certain feature may depend on the specific nonlinear optimization algorithm as well. The two most competitive NLP solvers are compared in more detail in the next section.

6.2.2 Nonlinear optimization solvers

From the nonlinear optimization packages that we tested during our research, the two solvers CONOPT and MINOS5 were the most competitive solvers for the continuous relaxations of the basic model. These two solvers are compared in Table 6.7. For all listed test problems, the computation time of CONOPT was smaller than the computation time of MINOS5. For the small problems, the quality of the results from both solvers is comparable. For larger problems, the solution quality of CONOPT is on average better. CONOPT is also more stable in the sense that it more often finds a feasible solution (in the test of Table 6.7 it always returns a feasible solution, but this is dependent on the model settings). In all other tests, we therefore restricted ourselves to the solver CONOPT. It should be noted that the CONOPT version used within GAMS is more recent than the callable library version that is used in the rounding algorithm used in this chapter (see Table 6.1), and we found that this older version performed slightly worse on our problems. We will come back to this in the discussion of Table 6.8.

Some other solvers gave much worse results and are only discussed in Chapter 7.

nr.	Small problems				Medium problems				Large problems			
	k^{eff}_{EoC}		time (s)		k^{eff}_{EoC}		time (s)		k^{eff}_{EoC}		time (s)	
	CPT	MN5	C	M	CPT	MN5	C	M	CPT	MN5	C	M
1	1.00802	**1.00958**$^+$	**1.6**	2.0	**1.03514**	1.03300	**26.8**	113.2	**1.04730**	1.04721$^+$	**191.5**	1229.8
2	1.04552	**1.04800**	**0.7**	2.0	**1.03440**	1.03413	**20.2**	112.0	**1.04731**	1.04700	**171.6**	1164.9
3	1.01976$^+$	**1.02021**$^+$	**1.2**	2.3	**1.03471**	1.03453	**18.8**	153.0	**1.04706**	1.04685$^+$	**166.0**	1539.3
4	**1.04273**$^+$	1.04043	**1.2**	3.8	**1.03385**$^+$	1.12457*	**13.3**	160.6	**1.04658**$^+$	1.04637	**143.8**	1060.2
5	**1.04009**	0.90094*	**0.7**	51.2	**1.03362**$^+$	0.81000*	**18.1**	155.4	**1.04638**	1.04623	**170.0**	1551.9
6	**1.03402**$^+$	1.03322$^+$	**0.9**	3.5	**1.03532**	1.03473	**18.7**	88.6	1.04740	**1.04744**	**159.4**	1283.4
7	1.02639	**1.02735**$^+$	**1.1**	2.5	**1.03354**	1.03340	**16.7**	121.7	**1.04665**	1.04655	**226.9**	1086.9
8	1.02950	**1.02955**$^+$	**1.0**	2.4	**1.03533**	1.03474	**23.5**	244.9	1.04707	**1.04725**	**175.8**	1248.4
9	**1.02683**	1.02624	**0.9**	2.7	**1.03372**	1.03349	**21.0**	140.9	**1.04661**$^+$	1.04647	**147.2**	1493.5
10	1.02470	**1.02486**	**0.9**	2.6	**1.03317**	1.03256	**31.7**	124.4	**1.04617**$^+$	1.04596	**148.7**	1463.9

*=No feasible solution found.
$^+$=Relaxed optimum is integer.

TABLE 6.7: CONOPT vs. MINOS5. Only the optimal solution value of the continuous optimization is reported. The test is performed by using the basic model within the modeling language GAMS.

6.2.3 Mixed-integer solvers

In Section 5.3, four different solution strategies were discussed for the integer part: rounding, outer approximation (OA), Branch-and-Bound and pairwise interchange (PI). Our quick and simple Branch-and-Bound implementation did not give useful results for the basic model, as is already discussed in Section 5.3.3. The other three algorithms are compared in the current section. The OA algorithm is tested using the package DICOPT, using the mixed-integer linear optimization (MILP) solver CPLEX (that performed much better on our problem than the solver ZOOM-XMP). We compare different settings of the CPLEX solver. Next, we compare different variants of our PI implementation. Finally, the different algorithms and their combinations are compared to each other.

DICOPT

When using DICOPT, it occurred quite frequently that the MILP subproblems took an extraordinary amount of time. In order to overcome that problem, we tested different CPLEX settings for node selection and branching. Since our nonlinear problem has many local optima, it may be expected that the linear problems have multiple integer solutions with roughly the same objective value. The standard best bound node selection may therefore stay a long time in Branch-and-Bound nodes at a high level. An alternative choice is the use of depth first node selection.

After choosing a node, a variable has to be selected. CPLEX contains a switch to recognize so-called Special Ordered Sets (SOS-variables) and to treat them in a dedicated way. Our binary variables are SOS-variables because all x-variables in a node have the property that exactly one of them is one, the others are zero, hence we tested this SOS switch.

The variable selection is followed by the choice whether to evaluate first the rounding up or down. In the default setting, CPLEX uses a heuristic to decide. Since we know that branching an assignment variable to one immediately fixes all other variables in the same row and column to zero, we tested whether forcing branch up speeds up the optimization.

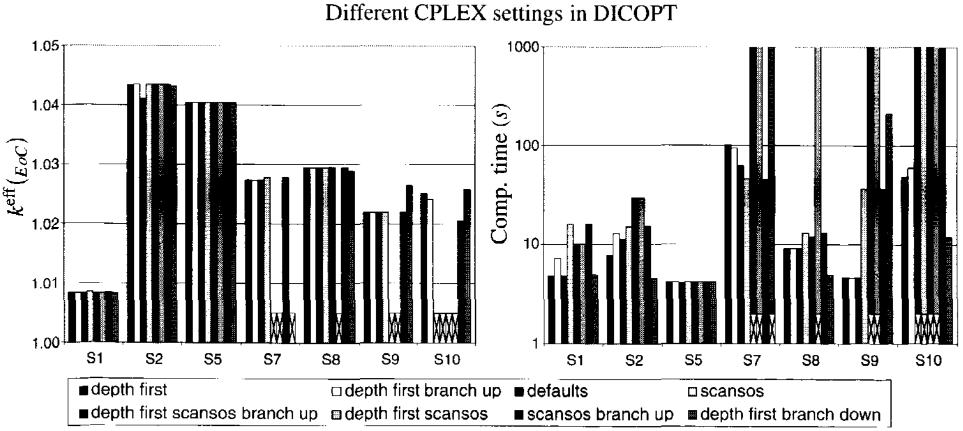

Different CPLEX settings in DICOPT

FIGURE 6.7: DICOPT, basic model, CONOPT for NLP, different CPLEX settings, small model. In model 3, 4 and 6 the NLP gave already an integer solution, and CPLEX was not invoked. A k^{eff}_{EoC}-value of 1.001 indicates that no integer solution was found, or the computation time exceeded 1100 seconds.

The results for different CPLEX solver settings are shown in Figure 6.7. Since these tests are time-consuming for the larger problems, results are only shown for the small test problems. As can be seen, the Depth first with default branching and Depth first combined with branch up, are the only two settings that lead to an integer solution in less than 1000 seconds in all cases, and there is very little difference between these two settings. In our further computations with DICOPT, we choose to use the Depth first setting with default branching.

Pairwise Interchange

In Figure 6.8 on page 143 two modifications of the standard PI are tested. The first one is our extension where the search continuous with the best-found infeasible solution as a parent, when no improving feasible pattern can be found. The second modification is the immediate selection of a new parent as soon as an improving solution is found. This last modification should be implemented with caution, since the order in which the neighbors are evaluated becomes important. We had all nodes in the core numbered from 1 to I, and all neighbors (swaps of two bundles) were evaluated in the order $(1,2),(1,3),\cdots(1,I),(2,3),\cdots,(2,I),\ \cdots,(I-1,I)$. Now suppose that the neighborhood starts with swap $(1,2)$ each time that an improvement is found and consequently the parent is updated. This means that there are lots of local improvements involving the lower-numbered bundles, leading to slow convergence. Each swap between two high-numbered nodes immediately causes a number of local swaps in the low-numbered bundles. As a better alternative we did not reset the current swap to $(1,2)$ after an improvement was made, but we continued with the next swap that was not yet evaluated, and after swap $(I-1,I)$ we continued with swap $(1,2)$.

By comparing the first and third algorithm from Figure 6.8 we see that it occasionally happens that immediate selection of a new parent leads to better solutions. Certainly this algorithm is much faster, although on average the solution quality is poorer. The algorithm with infeasible parent selection is by definition at least as good as the standard algorithm. For the small problems it gives much more improvement than for the larger problems. This can be explained because there are much less possible neighbors in the smaller problems, and also the influence of one exchange on the objective and on the feasibility is larger. The change of getting stuck in a local optimal solution for the standard problem is therefore much larger on the small problems. Also for the larger problems, the solution is often improved, with a reasonable increase of computation time.

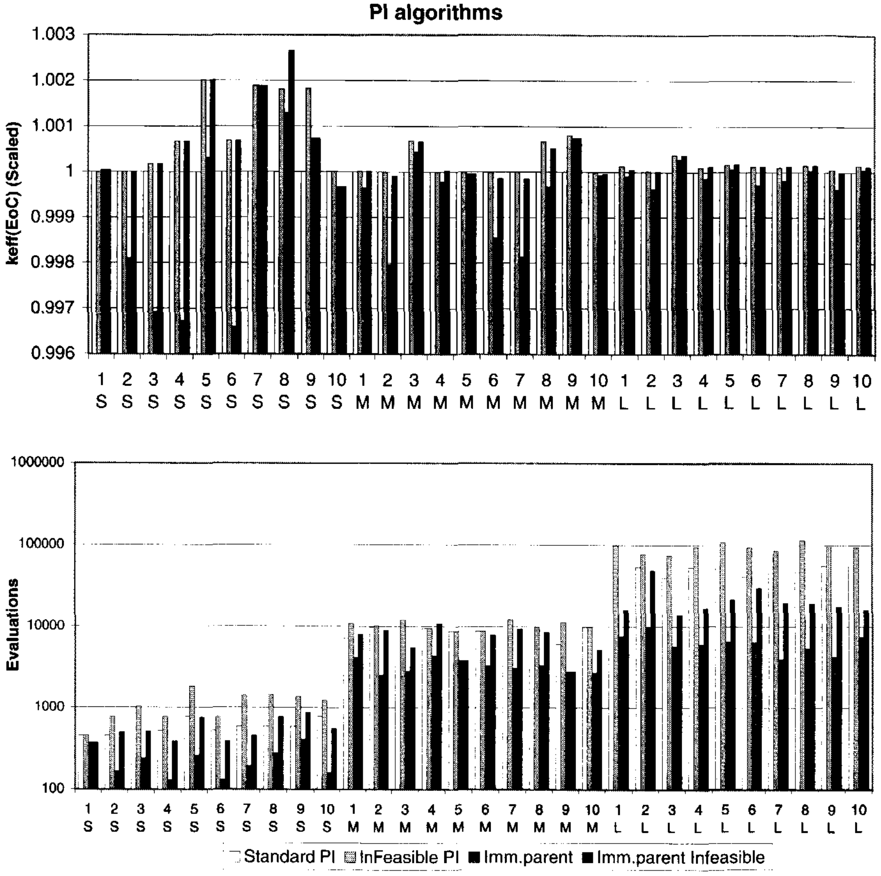


FIGURE 6.8: Results for different settings of the PI algorithm. The objective values of the first algorithm are scaled to 1, the others are scaled with respect to the first algorithm. Results are for the basic model with $T = 6$, forward discretization.

Comparing the different algorithms.

Six interesting algorithms are compared to each other in Table 6.8 on page 145. In this test, the best settings for the different algorithms that were found in the previous sections were used. Besides four basic algorithms we also use two algorithms (CRP and CRPR) where the best solution returned by the rounding algorithm is used as starting point for PI. This test was performed not only for the Small, Medium and Large test sets, but also for the Huge test sets. From these results we can make some observations:

1. The results of the relatively simple CR algorithm are competitive with the results from the more advanced DC algorithm. This may be due to the problem structure, where many NLP solutions are (almost) integer.
2. In some cases, the first NLP problem in DC returned immediately an integer solution (indicated with $^+$). One might spot that in such a case, CR should return the same value, since the first NLP is the same. The reason for this difference is that DC is performed under the modeling language GAMS, where we could use a newer version of CONOPT. Apparently this CONOPT version is slightly more robust for our problem.
3. The results of CR and DC without PI are on average slightly worse than PI results, but they are obtained much faster. Especially CR is very fast and robust in time. In five cases, it did not find a feasible integer solution, but in those cases it turned out that evaluating a few (PI)-neighbors lead to a feasible solution. For the Large and Huge cases it always returns a feasible solution in very moderate time. For these cases the problem becomes more 'continuous', change of one bundle has less influence, hence the objective and power peaking functions becomes more 'smooth', which improves the behaviour of the NLP algorithm.
4. Combination of CR with PI or PIR leads in all cases to improvement of the CR solution. The computation time is for the Small, Medium and Large test problems on average three-quarters of the solution time of the PI and PIR algorithm started from an arbitrary point.
5. PI with immediate parent selection after improvement (PIR resp. CRPR) is in almost all cases faster than a complete PI (PI resp. CRP), while the solution quality stays very reasonable.
6. Especially the CRPR algorithm has a good trade-off between on the one hand good quality solutions and on the other hand very reasonable solution times. Only for the Huge cases, it sometimes takes too much time.
7. The larger the problems, less difference in solution quality between the algorithms, and the larger the differences in solution time.

	k_{EoC}^{eff}						time (s)					
nr.	DC	CR	CRP	CRPR	PI	PIR	DC	CR	CRP	CRPR	PI	PIR
small 1	1.00832	1.01072	**1.01130**	**1.01130**	1.01114	1.01118	4.8	1.7	7.5	**7.4**	11.6	12.2
2	1.04330	1.04799	1.04844	1.04844	**1.04848**	1.04846	7.7	1.8	**6.6**	8.7	12.1	14.2
3	1.01976$^+$	1.01500	**1.02069**	**1.02069**	**1.02069**	**1.02069**	1.4	2.5	8.4	**6.6**	16.4	14.7
4	1.04273$^+$	1.04024	**1.04275**	**1.04275**	**1.04275**	**1.04275**	1.3	2.5	13.8	**7.3**	14.0	8.8
5	**1.04045**	1.03959	1.03974	1.03974	1.03974	1.03975	4.2	2.0	**5.3**	5.4	31.6	16.2
6	1.03402$^+$	1.03312	**1.03407**	**1.03407**	**1.03407**	**1.03407**	1.1	2.9	**7.2**	9.4	13.8	8.7
7	1.02739	1.02222	**1.02753**	1.02748	1.02682	1.02682	102.5	2.3	13.5	9.1	23.4	**8.3**
8	1.02947	1.02764	**1.02975**	1.02970	1.02888	**1.02975**	9.1	2.4	12.7	**11.3**	24.0	18.9
9	1.02198	1.02759*	**1.02703**	1.02698	**1.02703**	1.02591	4.6	3.5	13.4	**10.3**	21.6	17.8
10	1.02519	1.02490	**1.02545**	1.02530	1.02509	1.02475	47.3	2.2	**8.9**	9.0	19.1	11.2
medium 1	1.03514	1.03360	1.03532	1.03519	1.03545	**1.03546**	50.2	29.7	341.9	**154.9**	605.8	547.0
2	1.03441	1.03474	1.03528	1.03525	**1.03531**	1.03522	45.1	34.5	527.0	**215.3**	556.9	581.4
3	1.03470	1.03204*	1.03486	1.03489	**1.03493**	1.03490	45.3	47.1	411.7	**247.7**	666.6	347.7
4	1.03385$^+$	1.03396	1.03423	**1.03437**	1.03429	1.03433	15.4	35.3	**211.7**	232.6	516.8	651.7
5	1.03362$^+$	1.03264	1.03373	1.03373	**1.03380**	1.03376	20.7	32.4	310.5	**173.5**	455.0	231.7
6	1.03540	1.03494*	1.03555	1.03554	**1.03558**	1.03543	69.5	44.1	430.4	**184.1**	484.5	496.3
7	1.03357	1.03384*	1.03347	1.03395	**1.03411**	1.03396	42.0	43.9	460.3	**232.5**	644.3	685.5
8	1.03533	1.03446	1.03544	1.03535	**1.03549**	1.03533	48.0	25.5	544.2	**111.7**	553.7	544.2
9	1.03379	1.03370*	1.03350	**1.03418**	1.03413	1.03406	54.3	33.6	423.0	506.6	612.1	**174.2**
10	1.03317	1.03207	1.03330	**1.03339**	**1.03339**	1.03334	54.0	38.5	581.1	**181.7**	502.2	312.7
large 1	1.04729	1.04719	**1.04762**	1.04759	**1.04762**	1.04754	371.8	248.3	4107.8	3563.2	12105.2	**2008.6**
2	1.04731	1.04728	1.04753	**1.04754**	1.04751	1.04751	339.2	220.3	5281.8	**3585.2**	9516.9	6494.3
3	1.04704	1.04681	**1.04726**	1.04723	1.04724	1.04723	579.7	230.9	4769.4	**1463.2**	9234.5	1861.6
4	1.04658$^+$	1.04660	**1.04687**	1.04686	1.04684	1.04685	159.6	207.6	6825.0	**2451.3**	11240.8	2619.7
5	1.04638	1.04624	1.04651	**1.04652**	1.04650	1.04651	532.7	234.7	6217.0	4353.0	12282.3	**2787.8**
6	1.04735	1.04739	**1.04768**	1.04766	1.04767	1.04767	1070.3	259.4	8573.1	**3635.5**	11561.9	3887.8
7	1.04665	1.04645	1.04681	1.04680	1.04680	**1.04683**	384.9	188.7	7328.7	3601.6	9653.4	**2711.0**
8	1.04707	1.04697	**1.04763**	1.04760	1.04761	1.04761	296.6	241.8	10719.4	**1570.5**	14159.3	2495.0
9	1.04661$^+$	1.04663	**1.04677**	**1.04677**	1.04668	1.04668	159.5	219.1	4112.3	2642.7	13226.5	**2309.6**
10	1.04617$^+$	1.04601	**1.04627**	**1.04627**	1.04625	1.04622	165.7	232.1	8884.2	4413.8	10940.2	**2152.4**
huge 1		1.04436	**1.04452**	**1.04452**	**1.04452**	**1.04452**		1418.5	47377.8	**24628.4**	88776.6	26289.1
2		1.04429	1.04442	1.04443	**1.04448**	1.04446		1255.4	34242.0	**19548.4**	59937.9	19877.9
3		1.04410	1.04425	1.04426	**1.04431**	1.04428		1396.6	27822.7	27391.7	63868.8	**17128.4**
4		1.04378	1.04400	1.04395	**1.04401**	1.04400		1260.6	33299.2	**16801.4**	73687.6	25018.6
5		1.04350	**1.04378**	1.04373	1.04377	1.04370		1386.7	46296.1	**7328.7**	65931.6	23561.0
6		1.04441	1.04448	1.04450	**1.04459**	1.04452		1376.4	21771.9	**9174.4**	92012.3	34262.6
7		1.04369	1.04383	1.04390	**1.04396**	**1.04396**		1408.4	39347.0	30963.8	97230.8	**25647.1**
8		1.04435	**1.04454**	1.04442	**1.04454**	1.04448		1283.4	46838.3	**8006.1**	76082.6	27880.0
9		1.04372	1.04388	**1.04395**	**1.04395**	1.04392		1482.1	38839.8	**21569.1**	93739.8	25952.1
10		1.04331	1.04351	1.04355	1.04357	**1.04359**		1625.2	26509.2	**9503.0**	80643.1	27218.1

* = Feasible relaxed optimum found, but no feasible integer optimum found.

$^+$ = Relaxed optimum is immediately integer.

Boldface objective = Best objective value.

Underlined objective = Within 10^{-5} from best objective value.

Boldface time = Best time from the four rightmost columns.

Underlined time = Within 10% from the best time in the four rightmost columns.

TABLE 6.8: Results on the basic model with five different algorithms.

- DC = DICOPT, using CONOPT as NLP solver and CPLEX with depth first strategy as MILP solver;
- CR = CONOPT as NLP solver, followed by a rounding procedure;
- CRP = CR, followed by a PI algorithm (see below) starting from the rounding optimum;
- CRPR = CR, followed by a PIR algorithm (see below) starting from the rounding optimum;
- PI = Pairwise Interchange, in our improved form that explores the best infeasible pattern when no feasible iteration can be made;
- PIR = Pairwise Interchange where a new iteration is started as soon as a neighbor is computed with improved objective value.

6.2.4 Cuts

Different types of cuts are described in Section 5.4. In order to test the different cuts and combinations of cuts, we created a script around our program that generated random settings (within pre-specified ranges) for the different cut parameters that are specified in Section 5.5.2 on page 120. Based on experience with smaller test runs for each cut separately, upper and lower bounds on the α^i and ε^i were set for the different types of cuts. The parameters of the initial starting value (ω and starting pattern 1 or 2, see Section 5.1) were also varied in the different tests.

The tests were executed with the basic model for all 30 Small, Medium and Large test problems. For each test set, 200 test runs were performed. The results were compared to the results of the rounding algorithm without cuts, applied to the same basic model. The results show that in most of the cases, the cuts did not help in escaping from local optimal solutions. In many cases, either the cut algorithm could not escape from the attraction area of the previous local optimum, or it found a different local optimum with a worse objective value. Only in 10 from the 30 test sets, there were a few test runs where the cuts lead to a better optimal solution. The best of these solutions are given in Figure 6.9. The left bar is the solution of the basic model without cuts. The middle column is the best solution out of the 200 test runs of the cut algorithm, and the right bar is the solution obtained by PI. The solution values are scaled and are relative to the solution of the best cut solution. As can be seen, the extra time required for the cut algorithm compared to the rounding algorithm is limited. This is because the first NLP optimization without cuts takes much time, while the subsequent re-optimizations start already from a point that is relatively close to the new optimum. In three instances, the cut solution beats the PI solution with much more limited solution time.

Unfortunately, the results shown in Figure 6.9 for the different test sets were obtained by completely different parameter settings for the cuts. We were not able to explain these results. From a statistical view, a few vague relations could be found by viewing the correlation between the setting of one or two variables and the solution values obtained in the different test runs. This gave relations like:

- Settings where the sum of α^1 and ε^1 is around 0.2 - 0.3 gave in all but a few cases better results than settings where this sum was lower than 0.2 or above 0.3;
- The logarithmic variant of cut 2 gave on average better results than the linear variant;
- The most good results are obtained when $\alpha^2 \in [0.3, 0.4]$ and ε^2 is about $0.4 - \alpha^2$.

The general conclusion is however that it seldom pays off to implement the cuts for the basic model, although compared to PI, computation times are still very moderate and solutions are comparable or only slightly worse. These conclusions still tell nothing about the new fractional model, but testing was impossible due to time limitations. Since the objective function in the fractional model is much more smooth, it is to be expected that the attraction areas of the local maxima are larger, hence it may be more difficult to escape from them by simple addition of the cuts.

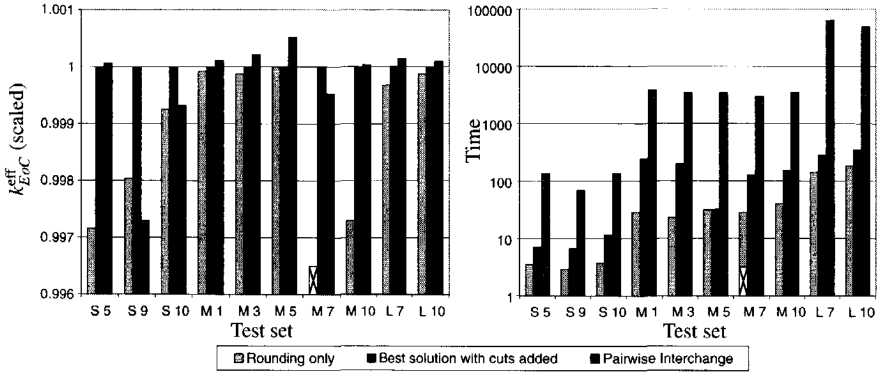


FIGURE 6.9: Results for Cuts.

6.3 Results for the BP model

The addition of cuts, the choice of the starting point, even the difference between forward and central discretization, are adjustments of the mathematical optimization model and algorithm of the same physical problem, with the only purpose of getting improved results for the same optimization problem. The addition of burnable poisons (BP) is an extension of the physical problem itself. The solution space of the optimization problem itself is enhanced, and if a better optimum is found by addition of BP, this is not a better local optimum of the basic model, but it is a solution that could never be obtained by using the basic model. If the BP model finds a better solution than the basic model, while no BP is actually used in this solution, then this is also a better local solution in the basic model. Another algorithm might have found this solution by the basic model as well.

We have implemented the BP model for the rounding algorithm only. It is not obvious at all how the addition of BP should be modeled in the PI algorithm. Since we use a model of continuously variable BP concentrations, the most logical way is to include BP optimization in a PI algorithm by performing a nonlinear optimization for fixed patterns to find the optimal BP distribution for these specific loading patterns. One possibility is to start such a NLP in each neighbor. This would lead to unacceptably large solution times. Moreover, the initial iterations are likely to be in a worse region of the solution space, where addition of BP still leads to solutions that are worse than the best solution in other regions of the search space, so that initially, these computations are wasted. Another choice is to do the PI without considering BP, and to do a separate BP optimization with NLP only for the final pattern. In this way, we may overlook other patterns that are not so good or infeasible without BP, but that would have a better objective value when BP was added. An intermediate choice (probably the most promising) would be to relax the power peaking constraints in the PI algorithm and run a NLP with the actual power peaking constraints to find an optimal BP concentration for the subsequent parent nodes only. This would again give a large increase in solution time, especially compared to the rounding algorithm. It is far more elegant to include BP optimization in mixed-integer nonlinear optimization, where it can be included in a natural manner.

When using the BP model, it is important to consider the length of the time steps. Since the rate of change of the BP concentration decreases very fast shortly after BoC, too long time steps may cause inaccuracy and instability in the results. Using the assumptions from Section 5.2.6, the requirement

$$\Delta_t \leq \frac{1}{\alpha^{\mathrm{P}}} \cdot \frac{\sum\limits_{i=1}^{I} V_i}{f^{\lim}} \tag{6.1}$$

is a minimum requirement to obtain at least stable solutions for patterns that satisfy the power peaking constraints. For the different test data sets, from Table 6.2, (6.1) leads to the following values:

$$\begin{array}{ll} \text{Small:} & \Delta_t \leq 23 \\ \text{Medium:} & \Delta_t \leq 17 \\ \text{Large:} & \Delta_t \leq 28 \\ \text{Huge:} & \Delta_t \leq 28 \end{array} \tag{6.2}$$

In order to obtain stable solutions for patterns that do exceed the power peaking constraints, the Δ_t-values must still be considerably smaller. Seeing the effective Δ_t-values in Table 6.9, it seems to be a reasonable choice to have central discretization with uneven grid points and about 12 time steps.

Type of	Number of time steps														
discretization	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Forward even	70.0	58.3	50.0	43.7	38.8	35.0	31.8	29.1	26.9	25.0	23.3	21.8	20.5	19.4	18.4
Forward uneven	35.0	29.1	25.0	21.8	19.4	17.5	15.9	14.5	13.4	12.5	11.6	10.9	10.2	9.7	9.2
Central even	35.0	29.1	25.0	21.8	19.4	17.5	15.9	14.5	13.4	12.5	11.6	10.9	10.2	9.7	9.2
Central uneven	17.5	14.5	12.5	10.9	9.7	8.7	7.9	7.2	6.7	6.2	5.8	5.4	5.1	4.8	4.6

TABLE 6.9: Effective value of Δ_t when using different types of discretization with different number of time steps. Unevenly distributed time steps are distributed following (5.28). Here the smallest Δ_t at BoC is given.

In Figure 6.10 the results are shown for the rounding algorithm. The first bar shows the result for the basic model. The second bar shows the result for the BP model. In this case, the relaxed NLP problem is solved for the BP model, followed by rounding, where in each rounded solution, an NLP is solved to find the optimal BP distribution to the rounded solution. The third algorithm starts with the solution of the relaxed NLP problem without use of BP. In this NLP, the power peak relaxation as used in all models (see Section 5.2.3) is not penalized in the objective, so that in the optimal solution the power peaking constraints may be slightly violated. Starting from this NLP solution, another NLP optimization is started. In this NLP model, the power peaking relaxation is penalized as usual, and BP is included. This second NLP is followed by a rounding procedure, with a nonlinear optimization in each rounded solution to optimize the BP concentration. In the graph with results, some bars are dashed; in those cases the optimal solution of the BP algorithm does not include BP's.

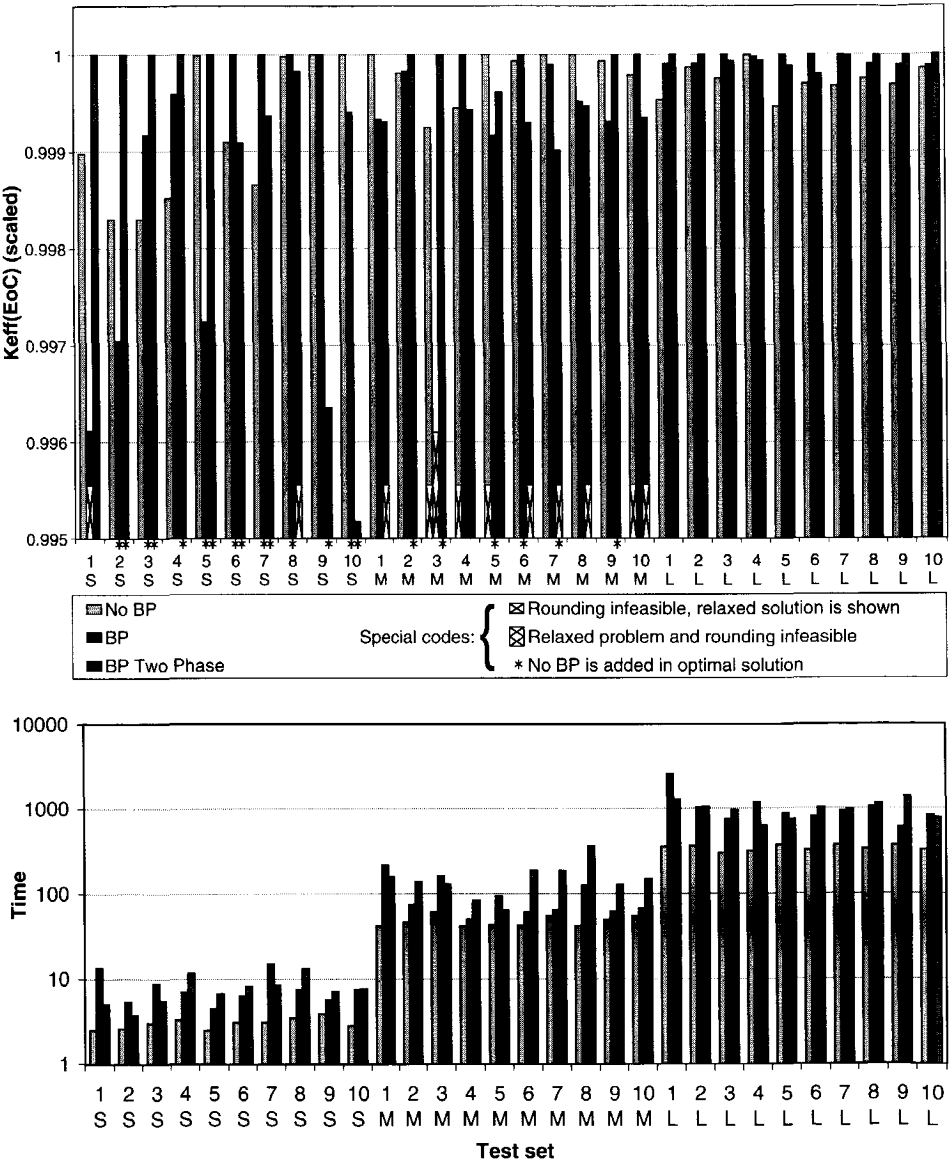


FIGURE 6.10: Results with and without use of BP.

The table shows some striking results. In one of the Small cases, three Medium cases and one Large case, the best integer solution is returned by the algorithm without BP. Clearly, the other algorithms got trapped into a local optimum. Another observation is that in relatively many cases, the third algorithm ends in a fractional solution with no feasible rounding. Apparently, the initial relaxation of the power peaking constraints leads to solutions where the power peak is so much violated, that it cannot be repaired by the addition of BP only.

A third striking result is that the BP algorithms give several times solutions where no BP is added, but that have a better objective value than the solution of the first algorithm. These improved solutions are also feasible solutions of the first model, which means that local optima are found that were not found by solving the basic model. This result is a bit in contrast with the earlier observation that the optimal solutions of the BP-algorithms are sometimes worse than the solutions of the original model. This may be explained by two forces that act against each other. On the one hand, the addition of BP leads to an increased feasible area. By adding BP in intermediate solutions, the BP model may reach solutions in some parts of the feasible region that could not be reached by the basic model. On the other hand, the increasing nonconvexity introduced by the BP equations may lead to an increasing number of local optimal solutions where the solver can get stuck.

The fourth, probably most important observation is that the solutions are more stable for the larger problems, and that for the large cores the BP models consistently outperform the results of the basic model without BP. Further, BP is used in all of these solutions. In the small test cores, the placement of one or two bundles has much influence on the maximal power peak and the objective function of the core. The violation of the power peaking constraints caused by a single bundle at a 'wrong' position is so strong that it cannot be corrected by addition of BP. For the larger cores, the placement of one bundle has much less influence on the total power distribution, and hence addition of BP may better damp out this power peak. These results also underline the observation that our MINLP approach proofs its real power on large size problems. The larger the problems are, the more stable are the results.

6.4 Conclusion

The results from this chapter show that it is not a trivial task to design an optimization algorithm for a complex problem like the loading pattern optimization problem. Neighborhood search is relatively simple, gives pretty good solutions but takes very much time. Moreover it cannot be extended to problems where continuous optimization is involved, like the continuous BP optimization[1]. The alternative that we investigated is using nonlinear optimization. From the result in this chapter, it turns out that in that case good modeling and careful implementation of the algorithms is a must. In Table 6.10 it is shown what we have gained with all modifications in this chapter, by comparing the result from the random starting point nr. 5 from Figure 6.3, the result of the CRPR algorithm, and the best result that we ever obtained by any algorithm for the different problems (translated to the 6-point forward discretized model). Actually, the first column is even not where we really started. This starting point has meaningful values for the

---

[1] We assume that the BP concentration can be chosen arbitrarily (within some bounds). This is not true for all types of BP. Some types of BP can be inserted in the fuel with integral quantities only. Treating them in one pass using PI will considerably enhance the solution space. We did not investigate the use of them in MINLP.

assignment variables, and the physical variables corresponding to this (fractional) assignment are computed using a fixed pattern evaluation algorithm. If these initial values were chosen at random, irregular things happened: we encountered a situation where a change in the fourth or fifth digit of one starting value could be the difference between finding no solution at all after a long time, and finding a good quality solution after a moderate number of iterations.

	Small problems			Medium problems			Large problems		
nr.	First	CRPR	Best-known	First	CRPR	Best-known	First	CRPR	Best-known
1	1.00676*	1.01130	1.01130	1.02979	1.03519	1.03546	1.04485	1.04759	1.04762
2	1.04201	1.04844	1.04860	1.02939	1.03525	1.03531	1.04488	1.04754	1.04754
3	1.00818	1.02069	1.02069	1.03023	1.03489	1.03493	1.04404	1.04723	1.04726
4	1.03527	1.04275	1.04275	1.02959*	1.03437	1.03437	1.04368	1.04686	1.04687
5	1.03373	1.03974	1.04045	1.02912	1.03373	1.03380	1.04349	1.04652	1.04652
6	1.02448	1.03407	1.03407	1.02886*	1.03554	1.03558	1.04486	1.04766	1.04768
7	1.02048	1.02748	1.02753	1.02984*	1.03395	1.03411	1.04395	1.04680	1.04683
8	1.02409*	1.02970	1.02975	1.03086	1.03535	1.03549	1.04461	1.04760	1.04763
9	1.02223	1.02698	1.02703	1.02998	1.03418	1.03418	1.04385	1.04677	1.04677
10	1.02039	1.02530	1.02573	1.02910	1.03339	1.03339	1.04334	1.04627	1.04627

*=Relaxed optimum found, but no integer feasible solution found.

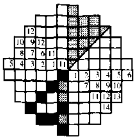
TABLE 6.10: Solution with which we started, solution of the CRPR algorithm and the best-known solution for the Small, Medium and Large test sets.

The results from this chapter showed that we finally arrived at algorithms that give good quality results in reasonable time. As could be expected, there is not one algorithm that gives always the best quality solution. From the results in Section 6.2.3 it however follows that our NLP+Rounding approach has much potential to give good solutions in moderate time, especially when it is combined with a 'post-processing' neighborhood search. In Section 6.3, dealing with the BP model, it is further shown that this approach is very flexible and that continuous optimization elements can easily be added. This is contrary to neighborhood search methods, that are essentially of discrete nature.

We conclude this chapter with a few words on the new fractional model. This model eliminates the very many local optima that make the optimization so difficult. In Table 6.6 it was shown that this potentially leads to better quality solutions. In the current stage of the research, a disadvantage of this model is that the computation time is much larger than with all other algorithms. There are two reasons for this that both require further investigation. The most important reason is the increasing number of variables and constraints, and the increasing number of nonzeros in the Hessian matrix of the nonlinear optimization model. For each node-bundle-timestep combination a new variable is introduced. In a BP model or a multi-nuclide model, such variables additionally have to be introduced for the BP concentration or for each separate nuclide, which makes the model very large.

The second (related) reason is the slow convergence. On the one hand, the objective function is more flat near the optimum, which makes it more difficult for the solver to reach the optimum. On the other hand most of the variables are zero in the optimal solution, since only a subset of all node-bundle combinations is actually used. Both features impede convergence.

More research is needed to tackle both problems. Regarding the explosive growth in the number of variables, one could think of some intermediate solution. Since only part of the variables is actually used, one might use some heuristic to decide which combinations are taken into account and which combinations are neglected. Regarding the problem of slow convergence, one might think of an interruption of the NLP solver as soon as convergence slows down, and then use a rounding technique to fix variables below some level to zero, and assignment variables above some other value to one. Once such a fixation is done, a large number of the node-bundle combinations – and hence a large number of the variables – can be fixed to zero. With such modifications, it will be very well possible that the fractional model combines both a very good solution quality and a reasonable computation time.



Chapter 7

Alternative approaches

In this chapter we discuss some solution methods for the fuel management optimization problem that differ from the ones discussed so far. Optimization via the world wide web and the use of the SIF input format gave some solutions for a few test problems, and are briefly discussed in Section 7.1 and Section 7.2, respectively. Some tests with global optimization algorithms are reported in Section 7.3. The use of semidefinite and copositive optimization to obtain good approximations, as discussed in Section 7.4, is only interesting from a theoretical viewpoint; practical application to fuel management is still far away. Concluding, Section 7.5 describes some preliminary work on parallel optimization. We implemented the PI algorithm on a parallel machine, leading to interesting results. Implementation of the other algorithms on a parallel machine is also discussed briefly.

7.1 Optimization via the web

A rather novel way of optimization is the use of an internet based optimization system. Via the NEOS server [91], one may send various types of optimization problems to a server on the web. These problems are then solved on a remote computer and the results are sent back. A few years ago it was possible to submit problems in SIF format (see next section) to be solved by the NLP solver LANCELOT and we did several tests using this interface. Currently, nonlinear optimization problems have to be submitted in the AMPL input format, a format that is somewhat similar to the more classical GAMS. At the time of writing this chapter, the following NLP solvers were available at the NEOS server.

- DONLP2: A combined SQP / Equality constrained QP method that is able to solve not too large problems (up to 300 unknowns and 700 constraints).
- LANCELOT: An Augmented Lagrangian Trust-region algorithm.
- LOQO: An Infeasible primal-dual interior-point method.
- MINOS: A Sequential Linearly Constrained optimization algorithm, where linearly constrained subproblems are solved by a reduced-gradient method with quasi-Newton approximations to the reduced Hessian.

- SNOPT: An SQP algorithm that uses a smooth augmented Lagrangian merit function.
- FILTER: An SQP algorithm.

Recently, also the mixed-integer solver MINLP (see the next Section) is added to the NEOS server. Due to time limitations we did not translate our models into AMPL format, and therefore we were not able to compare all these algorithms.

Another experimental project that enables remote solution of mathematical problems is the NetSolve project [16, 93]. This is a more general project that is not restricted to optimization problems. Currently, it provides access to several general numerical software libraries *e.g.*, for solving systems of nonlinear equations.

Web-based optimization offers opportunities for parallelization. It is possible to couple a number of workstations, whose actual location can be anywhere, provided that they are connected to the internet. Computational tasks may then be distributed over these machines. In a Pairwise Interchange algorithm for example, a large number of computers may calculate the solutions of the different neighbors of a parent. If every computer is finished, the best neighbor is selected as the new parent. Since most desktop computers in an average department do nothing during most of the time and are also connected to the internet, this opens an enormous potential of computing power. In order to use such a system efficiently, parallel algorithms are to be developed, since all computers have to do different tasks at the same time. More remarks about parallelization are postponed to Section 7.5.1.

7.2 LANCELOT and Leyffer's code

The solvers LANCELOT [19] and the MINLP solver of Leyffer [73] are treated in this section. Both solvers use the odd assembly-like language SIF as input format.

LANCELOT

We made an implementation of the basic model in this format, that enabled us to use the Augmented Lagrangian Trust-region solver LANCELOT. Since we could not obtain encouraging results, and most of the times no feasible solution was obtained at all, we contacted one of the authors of LANCELOT and sent our SIF files to him. After experimenting with lots of different settings of LANCELOT, it came up with a relaxed optimum for problem Medium 1 (about 1162 variables and 2312 constraints) with objective function 1.03420 in 10000 seconds on a reasonably fast machine. This should be compared to the results with CONOPT, that found for various settings solutions in the range 1.03440 to 1.03477 in less than a minute. As the author of LANCELOT agreed, the current version of LANCELOT is not competitive to CONOPT for this problem [128].

Leyffers' MINLP code

Independently, Leyffer converted the basic model into SIF format using his extension of this language, that allows for the specification of integer variables [74]. He has implemented his package MINLP containing a Branch-and-Bound method that uses an SQP solver to solve the continuous nonlinear subproblems [73]. He solved a modification of data set Small 2 (about 320 variables, of which 144 binary, and 624 constraints), for which he obtained an integer solution with objective value 1.02475 within a computation time of 361 seconds. In this time it solved 13 continuous NLP problems. A PI algorithm obtained an optimum value of 1.032344 for this case in about 15 seconds. CONOPT plus rounding obtained a solution with objective value 1.03122 in a few seconds. We may conclude that the Branch-and-Bound method of Leyffer offers a better alternative than LANCELOT for our problem, and at least it gives a convergent Branch-and-Bound implementation. In its current state, it is not yet competitive to our implementation that is based on CONOPT. It might be worthwhile to evaluate Leyffers' Branch-and-Bound method combined with the CONOPT NLP solver.

CUTE test set

SIF files of our basic model for different test sets are included in the CUTE standard optimization problem test set [22]. The implementation of Leyffer is available under the name C-Reload. Our implementation is available for the data sets Small 1, Medium 1 and Large 1 under the names TwiriSm1, TwiriMd1 and TwiriBg1, respectively.

7.3 Global optimization

For smaller-size problems, or for simple models, we considered the use of global optimization routines in order to generate guaranteed global optimal solutions. Three available codes are reviewed.

αBB

The code αBB of Floudas *et al.* [4] is based on a Branch-and-Bound method with over- and underestimates of the objective function and variables in each node. Overestimates are created by linear relaxations of the constraints, as well as by the addition of convex quadratic terms to nonconvex functions, in order to make the Hessian positive definite (Section 4.4.2). Underestimates are created using the NLP solver MINOS5. Tests were performed by Floudas using some variants of this code, and with different model reformulations on test problem Small 2 with forward differences and $T = 6$. The best result obtained with Pairwise Interchange and with MINLP to this problem has objective value 1.04848. The global optimization code αBB gave very slow convergence and finally returned a lower bound of 1.040244 and an upper bound of 1.187743. From experience, we expect the real optimum to have an objective value of about 1.05, so there is still a large gap. Some improvement can possibly be made since at the time that this implementation was made, we did not yet use the bounds tightening of Section 5.2.1. On the other hand, the results of a few iterations from a global optimization routine like αBB may help in further tightening the bounds on all variables. Using such improved bounds, one might obtain better results from a MINLP algorithm.

BARON

The second code is the BARON solver of Sahinidis *et al.* [111, 112]. This is also a Branch-and-Bound based method with further tightening of the bounds in each node. At the time being, no results of an implementation of test set Small 2 were available yet, however similar results as for αBB can be expected. A further development related to αBB is a special treatment of integer variables, recently developed by Tawarmalani *et al.* [124]. Since the placement constraints in our problem contain a lot of bi- and trilinear terms in which binary variables are involved, this technique may be helpful to get useful results for at least small instances of the loading pattern optimization problem.

LGO

The third code is the Lipschitzian optimization based code LGO of Pintér [101]. Contrary to the previous two codes, this is a real black box implementation. Only simple bound constraints have to be given explicitly to the code, all other constraint and objective values in sample points are obtained via a call to a user-written routine. Theoretically, this is a global optimization method in that it uses a Lipschitz based Branch-and-Bound method (Section 4.4.1). Practically, heuristics are used to keep the solution times reasonable, and global optimality of the solution is not guaranteed. We made an implementation in which only the assignment variables were explicitly used as variables; the other variables were hidden in the black box, and our fixed-core evaluation routine was used to determine the value of the objective function and the power peaking constraints for a given set of assignment variables. The strength of LGO -complete independence of the model and the solution method- is also its greatest weakness for the reload pattern optimization problem. Since the linear assignment constraints are treated by penalties and not used directly, the solver had difficulties in finding $x_{i,\ell,m}$-matrices that satisfied the assignment constraints. In our black box implementation the fixed core evaluation routine can only be invoked if the $x_{i,\ell,m}$-matrix satisfies the assignment constraints. This meant that during more than 99% of the running time, LGO was only searching for a valid assignment matrix, and finally returned a solution that was still not a valid assignment. Currently, the author of LGO is working on an improved version, that offers the possibility to treat linear constraints and integer restrictions explicitly instead of using penalties. This will probably lead to much more stable and relevant solutions.

7.4 Semidefinite and copositive relaxation

All constraints and the objective in our fuel management optimization models contain linear, bilinear and trilinear terms only. As is already shown in Section 5.2.5, the trilinear terms can be replaced by bilinear terms at the cost of introducing some additional variables. Bilinear terms can be relaxed using linearization, as is outlined in Section 4.4.2. Using such linearizations, one is able to find overestimates of the global optimal solution, that are used in the Branch and Reduce global optimization approaches. Another possible way to relax the bilinear terms is the use of convex conic optimization, which is the topic of this section. The best-known method in this area is semidefinite optimization (SDO), that is worked out in Section 7.4.1. In Section 7.4.2 we describe a copositive relaxation. This is a tighter conic relaxation that was developed as a by-product of our study. Both the semidefinite and the copositive relaxations are theoretically suitable to derive bounds on the fuel management optimization problem. In practice, currently available semidefinite optimization packages can only handle limited-sized problems (up to a few hundred variables), so they are only applicable to very small models and small core sizes. For the copositive relaxation it is even not clear at the moment whether an implementation is possible that will work well on problems of moderate sizes. We therefore only explain briefly the basic ideas of these algorithms. A more detailed discussion about semidefinite optimization can for example be found in a paper by Vandenberghe and Boyd [130], for a more detailed discussion about copositive optimization we refer to our paper [107].

7.4.1 Semidefinite Optimization

The bilinear constraints are translated into one constraint that some matrix should be positive semi-definite (PSD). This method of relaxing the problem is called the Shor relaxation, since it was first introduced by Shor [116] (see also [130]). Consider the General Quadratic Programming (GQP) problem without linear constraints

$$\begin{array}{lrcl} \max & x^T A_0 x, & & \\ \text{s.t.} & x^T A_i x & = & b_i, \quad i = 1, \cdots, m. \end{array} \tag{7.1}$$

Here $A_i,\ i = 0, \cdots, m$ are symmetric $n \times n$ matrices, $b_i \in \mathbb{R}$ and x is the variable vector of dimension n. This formulation is general enough to include quadratic constraints with linear terms [106]. This implies that (7.1) also includes quadratic inequality constraints, since these can be written as equality constraints by addition of a linear term with a slack variable. The General Quadratic Optimization problem (7.1) is a very general NP-hard problem [86], including, for example, integer programming[1] and optimization with general polynomial constraints (see, for example [108]).

In order to create a semidefinite relaxation, (7.1) is modified by putting all nonlinearities into one separate set of constraints. This is achieved by introducing the $n \times n$ variable matrix X, which is defined as the dyadic product

$$X = xx^T \tag{7.2}$$

The quadratic terms $x^T A_i x,\ i = 1, \cdots, m$ can be reduced to linear terms in X:

$$x^T A_i x = \operatorname{Tr}\left(x^T A_i x\right) = \operatorname{Tr}\left(A_i x x^T\right) = \operatorname{Tr}(A_i X) \tag{7.3}$$

where the trace $\operatorname{Tr}(X)$ of a matrix X is defined as the sum of its diagonal elements, so for two symmetric matrices X and Y,

$$\operatorname{Tr}(XY) = \sum_i \sum_j x_{ij} y_{ij}.$$

---

[1] A naive way to include the integrality constraints $x_i \in \{0, 1\}$ in (7.1) is to add the restriction $x_i(1 - x_i) = 0$.

Using (7.2) and (7.3), the following reformulation of (7.1) is obtained.

$$\begin{aligned} \max \quad & \operatorname{Tr}(A_0X), \\ \text{s.t.} \quad & \operatorname{Tr}(A_iX) = b_i, \ i=1,\cdots,m, \\ & X = xx^T. \end{aligned} \tag{7.4}$$

The only nonlinearity occurs in the relation $X = xx^T$. Semidefinite optimization replaces this equation by the weaker constraint

$$X \in \boldsymbol{S}$$

where $\boldsymbol{S}$ is the cone of *positive semidefinite (PSD)-matrices*, defined by

$$\boldsymbol{S} = \{S \in \boldsymbol{M} : x^TSx \geq 0 \ \forall x\} = \{S \in \boldsymbol{M} : S = \sum_{i=1}^{k} s^i s^{i^T}, k \geq 1\}, \tag{7.5}$$

where the s^i's are vectors of appropriate dimensions. As was already noted by Shor, linear inequality side-constraints should be quadratized in a special way because otherwise they will vanish. In [103], such a quadratization is worked out for linear equality constraints.

Contrary to (7.4), which is an NP-hard optimization problem, the semidefinite relaxation is solvable in polynomial time, like for example LP, and unlike NLP and MILP. Much progress was made in the last years in developing polynomial-time interior point methods for semidefinite optimization (see, *e.g.*, the reviews [109] and [130]). Still, there is currently some gap between theory and practice. Solvers for SDO problems are becoming available during the last years, but due to the unavoidable need for dense matrix computations, they are only efficient on small sized problems, and therefore cannot solve problems with more than a few hundred variables. Therefore, we only implemented a very much simplified fuel management optimization problem into one of the solvers, to see how tight upper bounds could be found. After some experimenting we found that solving these problems only resulted in trivial upper bounds.

7.4.2 Copositive Optimization

As is shown in our paper [107], one of the causes for the fact that semidefinite relaxation gives trivial upper bounds is the way in which linear side constraints are treated. In the current section, we illustrate the basic ideas of copositive optimization by addition of nonnegativity constraints to the GQP (7.1):

$$\begin{aligned} \max \quad & x^TA_0x, \\ \text{s.t.} \quad & x^TA_ix = b_i, \ i=1,\cdots,m, \\ & x \geq 0. \end{aligned}$$

Using the matrix notation from the previous section, this problem is written as

$$\begin{aligned} \max\quad & \mathrm{Tr}(A_0X), \\ \text{s.t.}\quad & \mathrm{Tr}(A_iX) \;=\; b_i,\;\; i=1,\cdots,m, \\ & X = xx^T, \\ & x \geq 0. \end{aligned} \tag{7.6}$$

When using semidefinite optimization, the last two sets of constraints of (7.6) are relaxed to the constraint

$$X \in \boldsymbol{S} \cap \boldsymbol{N}, \tag{7.7}$$

where $\boldsymbol{S}$ is the cone of semidefinite matrices (7.5) and $\boldsymbol{N} := \{N \in \boldsymbol{M} : N_{i,j} \geq 0,\ i,j = 1,...,n\}$ is the cone of nonnegative matrices. In copositive optimization, we introduce the cone of *completely positive matrices*:

$$\boldsymbol{B} \;= \{B \in \boldsymbol{M} : B = \sum_{i=1}^{k} s^i {s^i}^T,\ k \geq 1,\ s^i \geq 0,\ \ i=1,\cdots,k\} \tag{7.8}$$

and replace (7.7) by the constraint $X \in \boldsymbol{B}$. This cone is contained in $\boldsymbol{S}$ and also contained in $\boldsymbol{N}$, hence the relaxation is at least as tight as the Shor relaxation. In [107] it is shown that both relaxations are equal for problems with 4 or less variables. Using duality theory, an example can be given with 5 variables, for which the copositive relaxation is really tighter than the semidefinite relaxation.

Unlike semidefinite optimization problems, copositive optimization problems are generally not solvable in polynomial time. The reason is that the cones $\boldsymbol{S}$ and $\boldsymbol{N}$ have a relatively simple boundary, but the cone $\boldsymbol{B}$ is bounded by an exponential number of facets [84]. Although any convex conic optimization problem is solvable in a polynomial number of iterations [92], one iteration in copositive optimization will in its general form take an exponential number of computations. There is some work going on in order to find suitable and efficient algorithms for special cases, for example the case that X is restricted to be a tridiagonal matrix. Currently however, the main value of the algorithm is of theoretical nature.

7.5 Parallelization

Another side-step in our research is the use of a parallel computer. Using a parallel computer in the right way can speed up the solution of a problem, or in the same time a more thorough search can be performed. The notion of parallel computing can be extended to a network of ordinary computers, as is already mentioned in Section 7.1. In the current section, we first make some notes about the general concept of parallelization. Then we concentrate on parallelization in neighborhood search (we actually have an implementation with a parallelized PI), and finally we give some thoughts on possible parallelization in MINLP in Section 7.5.3. The current section is only of introductory nature. For a book on the use of parallel computers in mathematical optimization one may consult, *e.g.*, [17].

7.5.1 Some concepts in parallelization

In order to design efficient parallel algorithms and implementations, it is necessary to have some understanding of the basic concepts of parallel computers. Computers can be classified as follows.

SISD: Single Instruction Single Data. This is the 'classical' sequential computer: It executes one instruction at a time on one piece of data.

SIMD: Single Instruction Multiple Data. Multiple processors execute the same instruction in parallel on different data.

MIMD: Multiple Instruction Multiple Data. Multiple processors work independently from each other. They either operate on their own data, or on shared data in a way that avoids read or write conflicts.

The best-known application of SIMD is the *vector computer*. Usually, an operation on a piece of data is a 'pipeline' of several steps: Load the data from memory, Manipulate the data in one or more steps, and Store the data in memory. One operation has to go through the whole pipeline before another operation is started. A vector computer has special instructions that are used when the same operation has to be performed on a whole array of data. The next element of the array enters the pipeline immediately when the first element moves one step forward in the pipeline. Vectorization is implemented in many modern processors, and compiler programs should recognize simple loops or operations that can be vectorized.

Modern parallel computers are usually of the MIMD type. A number of processors can perform different tasks. The memory either may be shared by the processors, or each processor has its own memory (as is the case with distributed computing). This type of parallelization is most efficient if the processors can work independently each other. Several issues have to be addressed when one is making an implementation for such a parallel computer. Following [17] the most important things to care about are *Task partitioning*, *Task scheduling* and *Task synchronization*.

Task partitioning

The operations of the algorithm are to be divided into independent tasks that can be performed concurrently. A task in this sense is a part of the algorithm that operates on a part of the data. The tasks may be similar operations on different data sets. This holds for example for the evaluation of all neighbors in a pairwise interchange (PI) algorithm. The tasks may also be completely different operations, that can be computed concurrently because they do not depend on each other.

Task scheduling

The issue here is how to assign the tasks to one or more processors for simultaneous execution. In the PI algorithm, one may divide the total number of neighbors by the total number of processors, and assign at once an equal number of neighbors to each processor. Alternatively, one may assign the neighbors that are to be evaluated one at a time to each processor. Note that some scheduling algorithm has to be executed at run time on one of the processors, that assigns the tasks to the different processors.

Task synchronization.

The order of the execution of the tasks has to be specified and the data exchange between the tasks has to be synchronized to ensure correct progress of the algorithm. Suppose that at some stage in the PI algorithm the best known maximal solution value is 100, while processor 1 finds a solution with value 120 and processor 2 finds a solution with value 110. The following order of operations will give wrong answers and must be circumvented.

time slot	Processor 1	Processor 2
1	Read best-known solution $\leftarrow$ 100	$\cdots$ previous computations
2	$100 < 120 \rightarrow$ make update decision	Read best-known solution $\leftarrow$ 100
3	Update best-known solution $\rightarrow$ 120	$100 < 110 \rightarrow$ make update decision
4	next computations $\cdots$	Update best-known solution $\rightarrow$ 110

An error occurs because processor 2 can read the best-known solution between the read and write operation of processor 1. This situation is circumvented if processor 1 temporarily locks the best-known solution just before it reads the solution in time slot 1, and frees it after the update in time slot 3. In that case, processor 2 has to stay idle until processor 1 completes the operation, but the computations remain consistent.

7.5.2 Parallelization in neighborhood search

Since we made an implementation of our pairwise interchange (PI) algorithm on a parallel machine, we will describe this parallelization in some more detail. The Pairwise Interchange algorithm is almost perfectly suited for parallelization. As already stated, *task partitioning* is almost trivial, since the evaluations of the different neighbor solutions of the current parent solution can be done independently.

For *task scheduling*, there are several possibilities. One may assign one or a few neighbor solutions at a time to each processor. This causes a little overhead time since each processor has to ask the tasks scheduler for another neighbor each time it has finished a neighbor. As an alternative, one may immediately divide the number of neighbors by the number of processors, and assign the same number of neighbors to each processor. This may lead to an uneven load distribution over the different processors, since different neighbors may require different computation times. The optimal strategy may depend on the specific parallel environment. With the shared-memory 10-CPU parallel machine that we used for testing purposes, it worked perfectly well to assign a few neighbors a time to each processor (say 5, depending on the size of the test problem).

Task synchronization is possible in various ways. Processors may have local copies of the model parameters, so that they don't interfere when reading model parameters. Processors may update the common best-known solution each time they finish a neighbor. Since they have to lock the best-known solution during this operation, another processor might have to wait when it just started the same update, as explained in the previous section. Another option is that the processors maintain local copies of the best-known solution, while the shared best-known solution is only updated at the end of a main iteration. Since shared-memory access in our test computer was very fast, the processors compared the solution value of each neighbor immediately.

Besides the synchronization of the updates of the best-known solution, there is also the synchronization at the end of each main iteration. All processors should have finished the evaluation of the neighbors before the parent solution can be updated, and a new main iteration can only be started after a parent update.

Limited experience with the PI algorithm on the 10-CPU machine showed that especially for larger problems almost *linear speedup* was obtained, which is the maximal obtainable efficiency of parallel algorithms (see Figure 7.1). This means that nearly no processor time was wasted due to synchronization.

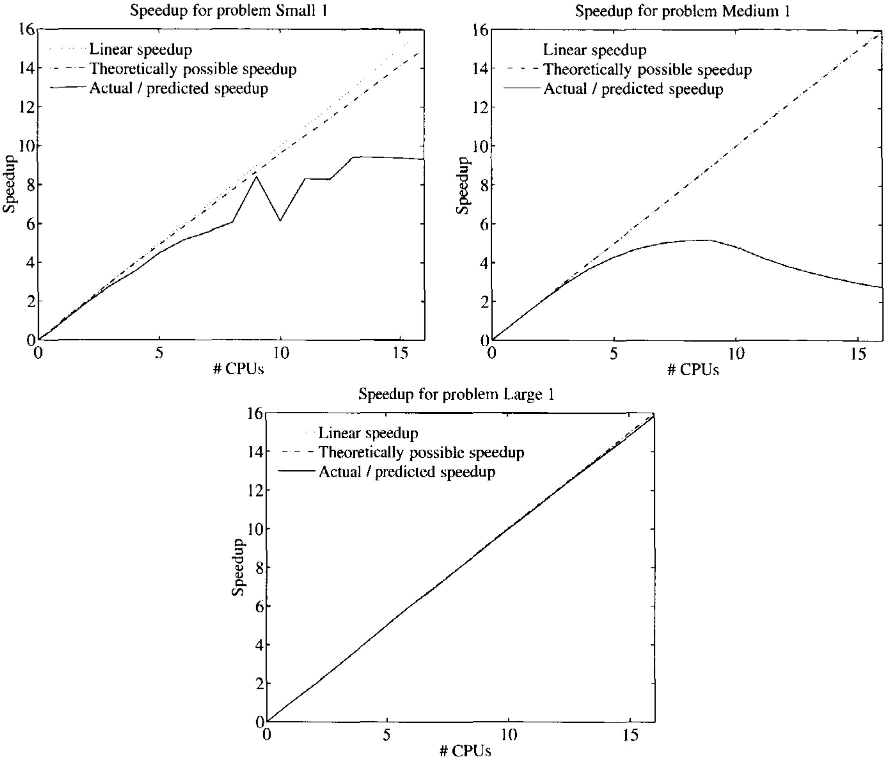
Also from the implementation side, the parallelization of PI is not too difficult. A compiler directive has to be added to the sequential Fortran code to tell the parallel compiler that the loop over all neighbors should be parallelized. Optionally, it can be specified how the tasks should be scheduled over the processors, this choice can also be left to the compiler. The directive also must contain information about which variables are used locally at each processor, and which variables are to be kept in shared memory. An additional compiler directive is needed to synchronize the use of the shared variables. Altogether, only about ten lines of compiler directives are to be added to the sequential code.

7.5.3 Parallelization in MINLP

The use of parallelization in mixed-integer nonlinear optimization is less clear. There are different possible choices in what should be parallelized, and the parallelizations may be far from trivial.

NLP

Let us first consider parallelization in continuous nonlinear optimization problem. This a rather unexplored area. There exist parallel versions of Quasi-Newton methods, but practical experience is very limited, and they may lead to the loose of sparsity structures [76]. A better approach for parallelization in nonlinear optimization seems to be the division of problems into several subproblems that can be solved separately. There is some experience on a parallel implementation of QP [35]. Nonlinear models that are separable, possibly up to a few global interconnecting constraints, may be parallelized using suitable decomposition techniques [76]. A general overview of the different possibilities of parallelization of continuous (non)linear programming is given in [17].


Speedup for problem Small 1

Speedup for problem Medium 1

Speedup for problem Large 1

FIGURE 7.1: Speedup that is achieved by parallelization of PI on the basic problems (see Table 6.4 for problem sizes). The dash-dotted line shows the maximum speedup that can be theoretically achieved, taking into account the part of the program that is run in parallel and the part of the program that is to be run in sequential mode. Only at the beginning and the end of the code, a sequential part is present; the evaluation of all neighbors is done in parallel. The effect of these sequential parts is reduced for larger problems, as can be seen by the fact that the dash-dotted line approaches the linear speedup line. The fixed line is the actual speedup for up to 10 CPUs, and an extrapolated speedup up to 16 CPUs. The behavior for smaller problems is very unpredictable due to several reasons:

- The evaluation of one neighbor is shorter, hence the chance that two processors have to synchronize their update of the best-known solution is larger;
- The number of neighbors is smaller, hence the computational task of one main iteration has to be divided into relatively larger parts. All processors have to wait at the end of each main iteration until the last one is finished, and the effect thereof is greater.
- Less data are obtained from a test run hence the extrapolation towards more processors is less accurate.

We do not have a satisfying explanation for the strange figure for the medium-sized problem. Besides the points just mentioned, some memory conflicts might have caused this result, that are only solvable when spending more effort in tuning the implementation.

MINLP

Besides the parallel implementation of each nonlinear optimization run, there are many more opportunities for optimization in mixed-integer nonlinear programming. In our rounding algorithm, all possible rounded loading patterns can be evaluated in parallel. In the outer approximation algorithm, the mixed-integer linear subproblems can be solved by parallel Branch-and-Bound (CPLEX has parallel versions for parallel machines).

Parallelization in a dedicated Branch-and-Bound algorithm is possible in several ways. The different branches from one parent node may be evaluated in parallel. In standard Branch-and-Bound, there are only two such branches. If more processors are available one may develop a dedicated Branch-and-Bound strategy where more branches are created. Given the binary variables $x_{i,\ell,m}$, with the constraints

$$\begin{aligned} \sum_{i=1}^{I} x_{i,\ell,m} &= 1 \\ \sum_{\ell=1}^{L} \sum_{m=1}^{M} x_{i,\ell,m} &= 1 \end{aligned}$$

one may create $L \cdot M$ branches, where for fixed i

$$x_{i,\ell,m} = 1$$

for all different pairs (ℓ, m). The different branches are evaluated in parallel.

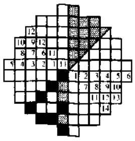
The descriptions thus far concentrate on reducing the wall-clock time for finding an optimal solution. Another view point is to use the same wall-clock time as in a sequential search, and to use the extra computing power to do a more thorough search. This is probably the biggest advantage of parallel computing. In this way, one may obtain solutions that could not be obtained by a sequential implementation, at least not in reasonable time. Since our algorithms are sensitive to the starting point, one may initiate a separate nonlinear optimization at each processor, each with a different starting point. Then one may choose the best resulting solution, or a small set of the best obtained solutions, and do a parallel evaluation of all roundings for these solutions. In the cut algorithm, one may create different warm starting points at the start of each cut iteration and re-optimize from these points in parallel.

The list of ideas for parallel implementation can easily be extended. Global optimization algorithms may also profit from parallelization. This especially holds for black-box type methods, where a number of core evaluations can be done simultaneously. Also Branch-and-Reduce methods may be parallelized, using the sketched ideas for Branch-and-Bound methods.

7.5.4 Conclusion

Parallel optimization opens perspectives to speed up optimization algorithms considerably. The time that is gained can be used to do a more thorough search, for example by using multiple starting points. In this way, one may obtain solutions that could not be found in reasonable time on a sequential computer.

The results of a parallel algorithm are very much dependent on the design and actual implementation of the algorithm, related to the number of processors and architecture of the parallel computing environment that is available. If one uses a parallel computer with only 10 processors, other parallel designs will be necessary than when a massively parallel computer with more than 1000 processors is used. The latter opens lots of possibilities, but it needs more effort to design an algorithm that is sufficiently parallel so that all 1000 processors are doing something useful all the time. Developing parallel algorithms will take much effort, and using computation time on a massively parallel computer is expensive. Still, seeing the savings in fuel cost that can be obtained by slightly improved loading patterns, it may pay off to invest more research in the development of parallel algorithms for the reload pattern optimization problem.



Chapter 8

Summary, Conclusions and Recommendations

The purpose of our research, as formulated in Section 1.3 on page 11 was

> *to explore the usefulness, to find the advantages and disadvantages, and to evaluate whether derivative based methods are potentially useful in loading pattern optimization, by developing suitable mathematical models and algorithms.*

In Section 8.1, we give a summary of the results that are collected in the intermediate chapters, and a summary of the conclusions that are drawn in these chapters. Section 8.2 then gives general conclusions with respect to the research goal. Section 8.3 lists a number of recommendations for further research.

8.1 Summary

Many nuclear power plants are up and running all over the world. In all these power plants a reload plan is made for every fuel cycle. The number of possible reload schemes for one fuel cycle is astronomical, and it is not possible to find the reload plan that is guaranteed to be the best one under the given objective. Instead, a reload pattern is sought that is expected to be good, with an objective value that is close to the theoretically best attainable one. Such reload patterns may be found using experience and the problem feeling of the designer, but usually sophisticated mathematical optimization techniques are needed to get really good reload patterns.

Most commonly, as we have seen in Chapter 3, heuristic techniques for discrete optimization are employed. Given a computer code that calculates the characteristics of a fixed loading pattern, these techniques generate and evaluate a series of patterns in a smart way, until a pattern is found that cannot be further improved by the used technique. Hopefully, such a pattern has an objective value that is not too far from the global best one. These techniques may be combined with expert knowledge to improve the results, or perturbation theory can be employed to fasten the search.

In this thesis, we have considered an alternative approach using gradient-based nonlinear (continuous) optimization techniques. Instead of doing core calculations for a large number of fixed trial patterns, all differential equations and related mathematical equations that describe the reload operation and the neutron behavior in the forthcoming cycle, as well as all operational constraints, are solved as one system of linear and nonlinear equations. An objective function is associated with this system of equations, and a solution of the system is found for which the objective function is optimal. This is done by using sophisticated techniques from the area of operations research, especially from mixed-integer nonlinear optimization (Chapter 4).

The approach based on nonlinear mixed-integer optimization to solve the in-core fuel management problem is not completely new, as we have seen in Section 3.2. Most of the results in literature however use either relatively simple algorithms, or use extremely simplified physical models. To our knowledge, the current thesis is the most complete study until now, combining sophisticated nonlinear optimization techniques and models that are realistic enough to give a quite accurate ranking of the loading patterns in terms of the objective value (Section 2.1.8).

During our research it became clear very soon that it is not sufficient at all to do just some standard implementation of a straightforward model using an arbitrary nonlinear mixed-integer optimization package. Instead, we found that the robustness, the solution time and the quality of the results were heavily influenced by the modeling, by the starting point supplied, by the nonlinear optimization algorithm and its implementation, by the mixed-integer optimization algorithm, and the tuning of algorithmic parameters.

Modeling

For the modeling, we found that the following items played an important role:

- *Tightening of variable bounds (Section 5.2.1):* using tighter variable bounds improved the ability of the nonlinear optimization algorithms to find feasible solutions, and to stay feasible during the optimization.
- *The addition of constraints (Section 5.2.2):* the addition of theoretically redundant linear constraints $k^{\text{eff}}_{t+1} \le k^{\text{eff}}_{t}$ helps the nonlinear optimization in the same way as the bounds.
- *Relaxation of constraints (Section 5.2.3):* power peak constraint relaxation helps a lot in obtaining feasible solutions, and to find good quality solutions.
- *The formulation of constraints:* the alternative reloading equation as shown in Section 5.2.5 is much more straightforward and simple than the tridiagonal one that we used, but it introduces pathological behavior for non-integer assignment variables.
- *The choice of variables:* the introduction of the k^{EoC} variables (Section 5.2.5) reduced significantly the computation time for some solvers (but not so much for others).
- *Choice of discretization:* a coarser discretization may lead to faster algorithms, but precision decreases and instability may be introduced. Also, a grid that is not equidistant may lead to a better compromise between precision and performance, as is discussed in Section 5.2.6.

- *The behavior of the model in fractional solutions:* the model that is used in most of the tests has many local peaks in the integer solutions. The new fractional model of Section 2.3.3 has a much smoother objective function – at the price of a much larger number of variables and nonzeros in the model.

- *Use of engineering constraints (Section 5.2.4):* though not used in our implementations, constraints from engineering knowledge may guide the search away from local optima. On the other hand they might disconnect the feasible area or make it otherwise more difficult to reach one part of the feasible region starting from another part.

Starting point

As is discussed in Section 5.1 and shown in Section 6.2, the choice of a starting point has major influence on the optimization procedure and on the obtained solution.

For the physical variables, meaningful starting points have to be supplied, otherwise the nonlinear optimization packages are often unable to solve the system of nonlinear equations, and no solution is found at all. This is not a problem, since a good approximation can be generated using iterative methods for fixed-core evaluation. If the assignment variables are chosen such that they represent a valid (possibly fractional) loading pattern, the fixed-core evaluation can be done for this loading pattern, resulting in a starting solution that is up to some precision already feasible for all constraints, except possibly the safety limitations (power peaking constraints).

The choice of the starting assignment is somewhat less critical than the choice of the physical variables, but still has very much influence on the quality of the solution obtained. Since the objective function is a nonconvex function of the assignment variables, starting in a bad part of the search space may result in a poor-quality local optimal solution. Here we used some engineering knowledge. In general, engineers have an idea of the general structure of a loading pattern. Loading patterns that are good with respect to our objective use to exhibit some ring structure. Our results show that it is important that the starting point contains somehow this ring structure, although there remains some freedom in the specification of this structure. In our basic model, it was important to generate non-integer starting points, since many integer solutions are local optima.

The choice of starting with a ring structure has a drawback, since nobody knows for sure whether there are possibly some other loading patterns, not having a ring structure, but having even better objective values. If one has some idea about the possible structure of such a pattern, it is a simple task to restart the algorithm with such a starting structure. The average result when starting in an arbitrary point, however, will be worse than when starting with a ring structure, so unless one has an idea of another good structure, starting with a ring structure is the best thing that can be done. This observation is valid for any known algorithm applied to the reload pattern optimization problem.

Nonlinear solver

Several nonlinear optimization packages that we have tested failed on many test sets to generate a feasible solution. After much testing, we found that the two solvers CONOPT and MINOS5 performed quite well on our problems. From the results presented in the thesis, we may conclude that CONOPT on average performed the best of all, both regarding solution quality and solution time. The relaxed solutions that were generated, with all modifications regarding the model formulation and starting point taken into account, have very reasonable solution values, that mostly compare favorably to the solutions found by other means. We have tried to further improve these results by embedding the nonlinear solver in a cut-generating algorithm, where problem-specific cuts are used (Section 5.4, Section 6.2.4). Unfortunately, our current implementation of those cuts seldom improved the solutions.

Mixed-integer solver

Complete enumeration of all possible roundings of the solution returned by the nonlinear solver works surprisingly well on our basic problems. This is due to the fact that the local optimal solutions are almost integer and contain only a few fractional assignment variables.

It is especially striking that this enumeration of rounded solutions works better than starting an *outer approximation* algorithm from the same solution of the nonlinear solver. Due to the severe nonlinearity of the model, the subsequent linearizations in the outer approximation algorithm redirect the solution to less favorable parts of the solution space. It is still an open question whether a dedicated problem-specific linearization in the Outer approximation algorithm would lead to better results than the standard straightforward linearization.

A straightforward application of the *Branch-and-Bound* principle is not suited for our problem, due to the nonconvexity. No conclusion can be drawn from a bad objective function high in the tree, since fixing a variable may redirect the search towards a better local optimal solution. A simple relaxation of the node fathoming rule by enlarging the tolerance on the objective value did not work: either the nodes are still fathomed too early, or the number of nodes to be explored grows too fast.

Our improvement of the *Pairwise Interchange* (PI) algorithm provides generally better solutions than standard PI, at the cost of a longer computation time (Section 6.2.3. An interesting effect is that it enables the PI algorithm with immediate restart after improvement, to generate solutions that are competitive with the standard PI, while the solution time is still smaller.

Alternative approaches

Until now, global optimization packages did not lead to interesting results. At the time of writing these conclusions, some work is still going on in this field on the reload pattern optimization problem, but the results until know are not that promising. Semidefinite and Copositive optimization are not yet suited for problems of the size of the reload pattern optimization problem. Parallel optimization, possibly using a network to create a virtual parallel computer, may enlarge the computer power available to solve the reactor problem. Much research is still needed to create powerful parallel MINLP algorithms.

8.2 General conclusions

In general, the results in this paper show that, with careful modeling and careful selection of tools, nonlinear mixed-integer optimization is a suitable tool to generate good quality loading patterns in relatively short time. Especially when combined with a simple and fast local search heuristic at the end, it provides good solutions in not too large time.

A disadvantage of the method at the current time is that it is *slightly less robust* than many heuristic methods. When the nonlinear optimization does not generate a feasible pattern as occasionally occurs, the solution is so close to a feasible one that a local search heuristic at the end will rapidly find a good quality feasible solution.

Another disadvantage may be that *advanced knowledge* of, and experience with nonlinear optimization modeling and algorithms is needed for the development of the models and the design and tuning of the algorithms. On the other hand, tuning a simulated annealing algorithm or an evolutionary algorithm is also a matter of experience.

An advantage of nonlinear mixed-integer optimization is its *speed.* Since it does not calculate the whole core behavior for a number of trial patterns, but instead used gradient information to move in the continuous search space, it reaches faster a local optimal solution than discrete search heuristics.

Another advantage, possibly the most striking one, is its *flexibility* because it allows to make and solve more realistic models. Extensions like Burnable Poisons or control rods can all be incorporated in the optimization model. Problem specific knowledge and engineering constraints are also easily added. Thereby, it makes in principle no difference whether all these extensions are continuous or discrete, although straightforward implementation of discrete or nonconvex extensions may decrease the robustness.

As a third advantage we may note that it can *easily be combined* with a search heuristic at the end. If one is not completely satisfied with the result, or if, due to a robustness problem, it is not possible to find a solution that satisfies exactly the safety constraints, then a few iterations by a search heuristic, starting from the point supplied by the nonlinear optimization, will generally lead to feasible, high quality loading patterns.

Generally, we may conclude that the approach described in this paper deserves to be explored further, to develop fast and stable dedicated algorithms that generate loading patterns of superior quality. A number of recommendations for this will be given in the next section.

From a broader perspective the results are also very interesting. It is already discussed in Section 1.2.3 that the reload pattern optimization problem belongs to a much broader class of problems, characterized by differential equations, where the parameters of these equations can be tuned to find good quality solutions. One might think of problems in engineering, construction, electronics, and a number of other application fields. Solving such problems using nonlinear (mixed-integer) optimization is a rather unexplored area. Our research shows that such an approach is feasible. The general methodology of combining knowledge about differential equations and knowledge of mathematical optimization may give powerful results.

8.3 Recommendations

Although we have examined and evaluated a lot of different methods on several models, the research was certainly not exhaustive. Extra testing is needed and some problems have to be resolved, and some possible improvements have to be investigated.

New fractional model

One of the most promising new developments during the last months of our research was the new fractional model. This model gives a better estimation of the objective function in case of fractional loading patterns. This leads to a more smooth objective function, and the results for the continuous optimization as shown in Table 6.6 are in many cases better than any result that was obtained with any discrete optimization algorithm. Still, some problems with respect to this model have to be resolved.

First of all, the number of variables and nonzeros grows very rapidly with the size of the core. Introduction of BP or a multi-nuclide model also leads to a fast growth of the number of variables of nonzeros. This slows down the nonlinear optimization algorithm and leads to large memory requirements. It should be investigated whether reduction of the number of variables is possible, for example by excluding node-bundle combinations that are likely to be very bad. Possibly, a plain model with very few time steps can be used to identify such combinations. It might also be investigated whether reduction of variables is possible by only counting separate node-bundle variables for the 2 or 3 largest fractions of a bundle. Furthermore, during a run of an optimization algorithm, it may be detected that some variables tend to zero, and then they may be fixed to zero in an early stage.

Another issue regarding the new fractional model is how to derive integer solutions. Contrary to the basic model used in this thesis, the fractional model gives fractional solutions that are not close to integer. The complete enumeration of simple roundings may be costly or even impossible, and does not necessarily generate good solutions. However, since the objective function is much more smooth, a dedicated Branch-and-Bound algorithm may work very well. It should also be considered whether the outer approximation algorithm can be applied using some dedicated linearization.

Cuts

The algorithms with cuts did not improve the results. This does not imply that cuts will never work. Cuts should be reconsidered and new types of cuts should be developed when the new fractional model is introduced. The idea of using cuts still seems to be reasonable, although we did not have enough time to work further on cutting or Branch and Cut algorithms.

Model extensions, model precision

Another point of study is the extension towards models that are more realistic. Hereby we think of the inclusion of control rods, more types of nuclides, *etc.* Robust and efficient models and algorithms are needed. It may be possible that a relatively simple model is used to identify promising regions in the search space, and only then a more detailed model is introduced to do a refined search. A related question that one should ask with every extension is, how much detail is still helpful in an optimization context. It may be that the core analysis is not completely accurate, but that the ordering of the loading patterns in terms of the objective function stays the same.

Benchmarking

This also leads to the question of how our method performs in comparison with all the other methods that are presented in literature. As already remarked in Section 3.4, a set of standard problems should be designed that is accepted by the nuclear engineering community, so that methods can really be compared with each other. Each problem in this test set should consist of a data set describing core geometry, available fuel bundles, and all necessary physical parameters, together with a detailed fixed-core evaluation model and a well-defined objective and operational constraints. A method that is to be tested then has to find a loading pattern with the best obtainable objective function, and a test report should list the pattern found, together with its objective function and the solution time, plus the starting point and other knowledge that is used. Using this information, the results should be reproducible when using the same implementation.

MINLP and Perturbation theory

One thing that we have not studied but that may have much potential is the combination of MINLP and perturbation theory. Nonlinear optimization needs the computation of gradients. Perturbation theory is a way of computing first order (and possibly second order) estimates of the effect when one or a few variables are changed. The two methods have much in common, and a collaboration between people with a mathematical optimization background, people with nuclear engineering background and people with thorough knowledge of numerical differential equations and perturbation theory, could possibly result in powerful algorithms, that may have application areas far beyond fuel management only. As already mentioned in the previous paragraph, many applications of optimization of differential equation parameters can be imagined, and use of both MINLP, and (momentarily still farther away) its possible combination with perturbation theory, is a field of research where much can be done.

Appendix A

Nonlinear optimization algorithms

This appendix presents a more detailed mathematical description of the constrained nonlinear optimization algorithms of Section 4.2.4. The basic optimization problem is

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \cdots, m, \\ & h_j(x) = 0, \quad j = 1, \cdots, p. \end{aligned} \tag{A.1}$$

The *Lagrange function* or Lagrangian is

$$L(x,u,w) := f(x) + \sum_{i=1}^{m} u_i g_i(x) + \sum_{j=1}^{p} w_j h_j(x), \tag{A.2}$$

which is defined for $u_i \geq 0$, $i = 1, \cdots, m$. The elements u and w are known as the *Lagrange multipliers*. In the sequel, we assume that the objective and constraint functions are continuously differentiable. We recall from Section 4.2.4 the Karush-Kuhn-Tucker conditions, that give first order necessary conditions for a point x to be a minimizer.

Theorem A.1 (KKT-conditions) *Given the optimization problem (A.1), where f, g_i, $i = 1, \cdots, m$ and h_j, $j = 1, \cdots, p$ are continuously differentiable. Let $\bar{x}$ be a point satisfying all constraints, and let $\nabla g_i(\bar{x})$, $i = 1, \cdots, m$ and $\nabla h_j(\bar{x})$, $j = 1, \cdots, p$ be linearly independent at $\bar{x}$. If $\bar{x}$ is a local optimum of (A.1) then there exist scalars $\bar{u}_i$, $i = 1, \cdots, m$ and $\bar{w}_j$, $j = 1, \cdots, p$ such that*

$$\begin{aligned} \nabla_x L(\bar{x}, \bar{u}, \bar{w}) &= 0, \\ \bar{u}_i g_i(\bar{x}) &= 0, \quad i = 1, \cdots, m, \\ \bar{u}_i &\geq 0, \quad i = 1, \cdots, m. \end{aligned} \tag{A.3}$$

A point $(\bar{x}, \bar{u}, \bar{w})$ satisfying (A.3) is called a KKT point. Basically, the different NLP algorithms aim at finding a KKT point, thereby risking that they sometimes end in a maximum, a saddle point or an inflection point, as illustrated in Section 4.2.1.

In most constrained optimization algorithms, given a current solution vector x^k, an iteration k is basically comprised of the following steps.

1. *Convergence test.* If the convergence criteria are satisfied, the algorithm stops with solution x^k.
2. *Search direction.* A nonzero vector s^k representing the search direction is calculated.
3. *Line search.* A certain positive step length λ is calculated, such that $f(x^k+\lambda s^k) < f(x^k)$, and the point $x^k+\lambda s^k$ is feasible.
4. *Update the current point.* $x^{k+1} = x^k + \lambda s^k$, $k = k+1$, and the algorithm returns to step 1.

   In steps 3. and 4. the term $x^k+\lambda s^k$ may be augmented by an additional term $\rho^k(\lambda)$ in order to maintain or restore feasibility.

This scheme is almost the same as the description of an iteration in an unconstrained optimization algorithm as is shown in Section 4.2.2, but in the actual implementation there are a number of differences. The calculation of the search direction is often performed by solving another, simpler constrained or unconstrained optimization model. Some major solution algorithms are discussed below.

A.1 Overview of algorithms

In most of the descriptions below, we concentrate on step 2 of the algorithmic scheme as sketched above. Methods for Line search (step 3) are discussed in Section A.2.2.

A.1.1 Sequential or Successive Linear Programming (SLP)

The search direction is computed by solving a linearization of the nonlinear problem around the current solution x^k:

$$\begin{array}{ll} \text{minimize} & f(x^k) + \nabla f(x^k)^T(x-x^k) \\ \text{subject to} & g_i(x^k) + \nabla g_i(x^k)^T(x-x^k) \le 0, \;\; i = 1,\cdots,m, \\ & h_j(x^k) + \nabla h_j(x^k)^T(x-x^k) = 0, \;\; j = 1,\cdots,p, \\ & -\Delta^k \le x - x^k \le \Delta^k. \end{array}$$

Here, Δ^k is a small positive number or vector, introduced since the linearization is only valid on a local scale. An equivalent formulation is

$$\begin{array}{ll} \text{minimize} & \nabla f(x^k)^T s^k \\ \text{subject to} & g_i(x^k) + \nabla g_i(x^k)^T s^k \le 0, \;\; i = 1,\cdots,m, \\ & h_j(x^k) + \nabla h_j(x^k)^T s^k = 0, \;\; j = 1,\cdots,p, \\ & -\Delta^k \le s^k \le \Delta^k. \end{array} \tag{A.4}$$

The line search parameter λ may be set to 1, or is determined by the techniques described in Section A.2.2. The value Δ^k can be constant during subsequent iterations, or it can be adjusted based on the quality of the linear estimation in the previous iteration (see also Section A.1.7 about Trust Region methods).

Once a linear solver is available, SLP is extremely simple. The disadvantage is its relatively slow convergence.

A.1.2 Sequential Quadratic Programming (SQP)

This is an enhancement of the SLP method. The search direction is computed by solving an approximation of the problem with a quadratic objective function, subject to the linearized constraints:

$$\begin{aligned} \text{minimize} \quad & \nabla f(x^k)^T s^k + \tfrac{1}{2} s^{k^T} H s^k \\ \text{subject to} \quad & g_i(x^k) + \nabla g_i(x^k)^T s^k \leq 0, \quad i = 1, \cdots, m, \\ & h_j(x^k) + \nabla h_j(x^k)^T s^k = 0, \quad j = 1, \cdots, p, \\ & -\Delta^k \leq s^k \leq \Delta^k. \end{aligned} \tag{A.5}$$

In the basic form, H is the Hessian of the objective function. Commonly, the name SQP is used for the Projected Lagrangian Method, discussed in the next section, in which H is the Hessian of the Lagrange function. The Hessian matrix may be specified by the modeler, or approximated by finite differences, but usually it is estimated by a Quasi-Newton algorithm, as discussed in Section A.2.1. Since a Quasi-Newton method can be forced to generate positive semi-definite approximations of the Hessian, fast dedicated algorithms to solve the quadratic subproblems (A.5) can be used.

A.1.3 Projected Lagrangian Method (PLM)

As in SQP, the PLM solves a linearly constrained subproblem to find the search direction. The objective function is related to the Lagrange function. In one variant, the Lagrange function itself is minimized, subject to linearized constraints. It can be argued (see for example [45, p. 234]) that this is not a good choice. A better choice is to minimize a quadratic approximation of the Lagrange function, where the gradient terms related to the constraints are removed (they are stated explicitly in the linearized constraints). With this objective function, PLM is an extension of SQP where the Hessian is taken to be the second derivative with respect to the x-variables of the Lagrange function (A.2). In order to use the Lagrangian, not only an estimate x^k has to be available at the start of the kth iteration, but also estimates of the Lagrange multipliers u^k and w^k must be available. The search direction is then computed by solving the following minimization problem.

$$\begin{aligned} \text{minimize} \quad & \nabla f(x^k)^T s^k + \tfrac{1}{2} s^{k^T} \nabla^2_{xx} L(x^k, u^k, w^k) s^k \\ \text{subject to} \quad & g_i(x^k) + \nabla g_i(x^k)^T s^k \leq 0, \quad i = 1, \cdots, m, \\ & h_j(x^k) + \nabla h_j(x^k)^T s^k = 0, \quad j = 1, \cdots, p, \\ & -\Delta^k \leq s^k \leq \Delta^k. \end{aligned} \tag{A.6}$$

The Lagrange multipliers u^{k+1} and w^{k+1} are updated to be the Lagrange multipliers associated to the constraints of problem (A.6) in the optimal solution.

A.1.4 Augmented Lagrangian method (ALM)

The Lagrange function for a general nonlinear optimization problem is usually not convex. A minimum of the constrained optimization problem is even with the right Lagrange multipliers not necessarily a minimum of the Lagrangian, but it can also be a saddle point or (for fixed Lagrange multipliers) a maximum. One way to improve the PLM is addition of a quadratic penalty term for the equality constraints to the Lagrange function. So we solve the following problem at the kth iteration:

$$\begin{aligned} \text{minimize} \quad & L(x, u^k, w^k) + \frac{\theta}{2} \sum_{j=1}^{p} h_j^2(x), \\ & -\Delta^k \leq x - x^k \leq \Delta^k. \end{aligned} \tag{A.7}$$

This problem can be solved by an algorithm for unconstrained optimization, where the line search is restricted by the bound constraints. Extension of the augmented Lagrangian to inequality constraints can be done by addition of slack variables ([9, p.384]) or by maintaining a list of active constraints at x^k ([45, p.231]).

An important conceptual difference between the PLM and the ALM is the unconstrained minimization in ALM. In practical optimization, (A.7) is often not solved to optimality when the multipliers u^k, w^k and the parameter θ are updated. If the updates are done after each iteration of an unconstrained optimization method, the ALM becomes nearly equivalent to the PLM method ([45, p.232]).

A.1.5 Reduced Gradient method (RGM)

The RGM solves problems with nonlinear objective function and linear constraints. It is based on the ideas of the linear simplex method. For simplicity we assume that only equality restrictions and nonnegativity constraints are present:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0. \end{aligned} \tag{A.8}$$

The matrix A has size $p \times n$, where $p \leq n$, and the rows of A are assumed to be independent.

Before continuing, we need to define the concept of a basis and a basic feasible solution. If the feasible region of (A.8) is nonempty, each nondegenerate vertex of this region is defined by the set of p equality constraints, plus $n-p$ inequalities that are active. This means that $n-p$ variables are zero, and that the other variables are uniquely determined by the equality constraints. The $n-p$ variables at zero are called *non-basic* variables, and the dependent variables are the *basic* variables. Suppose that the vector of variables is ordered such that the set of basic variables, denoted by x_B, comes first, followed by the set of non-basic variables, denoted by x_N. The set of equalities in (A.8) can then be written as

$$(B\,N)\begin{pmatrix} x_B \\ x_N \end{pmatrix} = b, \tag{A.9}$$

where B is a $p \times p$ matrix, called a *basis*. The solution (x_B^T, x_N^T) is a *basic solution* for (A.8). If a *linear* optimization problem has an optimal solution, it always has an optimal basic solution. For a problem with linear constraints and a nonlinear objective function, this need not to be the case. Still any solution can be partitioned into parts (x_B^T, x_N^T) such that $x_B > 0$, $x_N \geq 0$ and the columns in the matrix A corresponding to x_B form a basis.

At the start of an iteration of the RGM a basis B is available, together with a feasible solution $((x_B^k)^T, (x_N^k)^T)$ satisfying (A.9), such that $x_B^k > 0$. Such a solution can e.g. be obtained by Phase I of the linear simplex method. A search direction (s_B^k, s_N^k) must be found such that[1]

$$\begin{aligned} Bs_B^k + Ns_N^k &= 0 && \text{(A.10)} \\ \nabla_B f(x^k)^T s_B^k + \nabla_N f(x^k)^T s_N^k &\lneqq 0 && \text{(A.11)} \\ s_j^k &\geq 0 \text{ if } x_j^k = 0. && \text{(A.12)} \end{aligned}$$

Here, $\nabla_B f(x^k)$ and $\nabla_N f(x^k)$ denote the gradient of f with respect to the basic and non-basic variables, respectively. Since B is invertible, we may rewrite (A.10) as

$$s_B^k = -B^{-1} N s_N^k. \tag{A.13}$$

This can be used to eliminate s_B^k from (A.11):

$$(-\nabla_B f(x^k)^T B^{-1} N + \nabla_N f(x^k)^T) s_N^k := r^T s_N^k \lneqq 0. \tag{A.14}$$

The gradient r is called the *reduced gradient*. The computation of a search direction reduces to the computation of a vector s_N^k satisfying (A.14) and (A.12). In a steepest-descent like setting, this can be done by choosing the elements of s_N^k such that $s_j^k = -r_j$ if $r_j \leq 0$, and $s_j^k = -x_j r_j$ if $r_j > 0$, *cf.* [9, p. 459]. There it is proved that this algorithm will give $s^k = 0$ if and only if x is a KKT-point. As an alternative to a steepest-descent like search direction, a Quasi-Newton type update may also be applied [45, p. 221] (also applied in the linearly constrained - nonlinear objective subsolvers of CONOPT and MINOS5). The computation of the search direction is completed by computing s_B^k from s_N^k using (A.13).

---

[1] The sign $\lneqq$ for a vector-inequality is used to denote that at least one of the inequalities must be strict.

Finally a line search has to be performed in the original search space. Since the direction s^k is by construction in the null-space of A, this line search is restricted by the nonnegativity constraints only:

$$\begin{aligned} &\text{minimize} \quad f(x^k + \lambda s^k) \\ &\text{subject to} \quad 0 \leq \lambda \leq \begin{cases} \displaystyle\min_{1 \leq j \leq n : s_j^k < 0} \frac{-x_j^k}{s_j^k} & \text{if } s^k \ngeq 0 \\ \infty & \text{if } s^k \geq 0 \end{cases} \end{aligned}$$

After computing the step length and x^{k+1}, the basis must possibly be changed. An easy rule is to sort the variables in decreasing order, and then, starting from the largest variable, enter p independent coefficient vectors into the basis for the next iteration.

Linear inequality constraints can be treated in the RGM in two different ways. One option is to make them equalities by the addition of slack variables. The other way is to use an active set strategy. This is a more complex method, but can be considerably faster for large problems.

A.1.6 Generalized Reduced Gradient (GRG) method

The standard RGM is applicable when only linear constraints are present. The GRG method is an extension to nonlinear constraints. As for the RGM, we treat only equality and nonnegativity constraints:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & h_j(x) = 0, \quad j = 1, \cdots, p, \\ & x \geq 0. \end{aligned}$$

Given a feasible solution x^k, the constraints are linearized around x^k:

$$h_j(x^k) + \nabla h_j(x^k)^T (x - x^k) = 0, \quad j = 1, \cdots, p.$$

If the constant terms are placed on the right hand side:

$$\nabla h_j(x^k)^T x = \nabla h_j(x^k)^T x^k - h_j(x^k), \quad j = 1, \cdots, p,$$

this can be viewed as a linear system of equations of the form $Ax = b$ and the search direction s^k can be found in the same way as in the RGM. Due to the linearization of the constraints, s^k may point outward the feasible set. This is corrected by adding an extra term:

$$x^{k+1} = x^k + \lambda s^k + \rho^k(\lambda),$$

where $\rho^k(\lambda)$ is orthogonal to s^k. For a given λ, $\rho^k(\lambda)$ is found by iteratively solving the nonlinear system of equations

$$h_j(x^k + \lambda s^k + \rho^k(\lambda)) = 0, \quad j = 1, \cdots, p$$

Finally a line search has to be performed in the original search space. Since the direction s^k is by construction in the null-space of A, this line search is restricted by the nonnegativity constraints only:

$$\begin{array}{ll} \text{minimize} & f(x^k + \lambda s^k) \\ \text{subject to} & 0 \leq \lambda \leq \begin{cases} \displaystyle\min_{1 \leq j \leq n: s_j^k < 0} \frac{-x_j^k}{s_j^k} & \text{if } s^k \not\geq 0 \\ \infty & \text{if } s^k \geq 0 \end{cases} \end{array}$$

After computing the step length and x^{k+1}, the basis must possibly be changed. An easy rule is to sort the variables in decreasing order, and then, starting from the largest variable, enter p independent coefficient vectors into the basis for the next iteration.

Linear inequality constraints can be treated in the RGM in two different ways. One option is to make them equalities by the addition of slack variables. The other way is to use an active set strategy. This is a more complex method, but can be considerably faster for large problems.

A.1.6 Generalized Reduced Gradient (GRG) method

The standard RGM is applicable when only linear constraints are present. The GRG method is an extension to nonlinear constraints. As for the RGM, we treat only equality and nonnegativity constraints:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & h_j(x) = 0, \quad j = 1, \cdots, p, \\ & x \geq 0. \end{array}$$

Given a feasible solution x^k, the constraints are linearized around x^k:

$$h_j(x^k) + \nabla h_j(x^k)^T (x - x^k) = 0, \quad j = 1, \cdots, p.$$

If the constant terms are placed on the right hand side:

$$\nabla h_j(x^k)^T x = \nabla h_j(x^k)^T x^k - h_j(x^k), \quad j = 1, \cdots, p,$$

this can be viewed as a linear system of equations of the form $Ax = b$ and the search direction s^k can be found in the same way as in the RGM. Due to the linearization of the constraints, s^k may point outward the feasible set. This is corrected by adding an extra term:

$$x^{k+1} = x^k + \lambda s^k + \rho^k(\lambda),$$

where $\rho^k(\lambda)$ is orthogonal to s^k. For a given λ, $\rho^k(\lambda)$ is found by iteratively solving the nonlinear system of equations

$$h_j(x^k + \lambda s^k + \rho^k(\lambda)) = 0, \quad j = 1, \cdots, p$$

by using a Newton-Raphson process. It may be that during this correction step one of the basic variables tends to become negative. In that case a change of the basis is needed. For more details about the computation of $\rho^k(\lambda)$ one may consult *e.g.*, [45, p. 222].

The correction $\rho^k(\lambda)$ may be computed only after the line search is finished, or at each trial solution in the line search. The latter case is computationally more expensive, but is more robust.

The GRG method is more complex than all previous methods, but its greatest advantage is that it maintains feasibility once a feasible solution is found. This is unlike the previous methods, where complete subproblems are solved to optimality with one set of linearized constraints. In that case the objective may be evaluated in points that satisfy the linearized constraints, but that not necessarily satisfy the real constraints. In the GRG method, if the correction $\rho^k(\lambda)$ is computed at each trial solution in the line search, the objective function only has to be evaluated in feasible points.

A.1.7 Augmented-Lagrangian Trust-region method (ALTR)

Trust-region methods differ from all previous methods because they do not have a separate line search after the computation of the search direction. Instead, the search direction depends on a trust region like the box constraints Δ^k in SLP. Let

$$L_A(x,u,w) = L(x,u,w) + \frac{\theta}{2}\sum_{j=1}^{p} h_j^2(x)$$

be the augmented Lagrangian of our problem, with gradient vector g^k and Hessian matrix H^k at a given point (x^k,u^k,w^k), where the gradient and Hessian are calculated with respect to x only. Then the following problem is solved in iteration k:

$$\begin{aligned} \text{minimize} \quad & (g^k)^T s^k + (s^k)^T H s^k \\ \text{subject to} \quad & ||s^k|| \leq \Delta, \end{aligned} \tag{A.15}$$

where $||\cdot||$ is usually the 2-norm or the ∞-norm, and Δ is some positive parameter. The solution is the same as the solution of the unconstrained minimization problem

$$\text{minimize} \quad (g^k)^T s^k + (s^k)^T (H + \alpha I) s^k,$$

where α is an appropriately chosen parameter. If $\alpha = 0$, then s^k is the unconstrained Newton direction in (A.15). If $\Delta \to 0$ then $\alpha \Rightarrow \infty$. In that case $(H + \alpha I) \to \alpha I$, and s^k approaches the steepest descent direction (although its norm tends to zero).

The solution s^k of (A.15) defines the *Cauchy point* $x^k + s^k$. Once it is obtained, the real reduction of the objective function $f(x^k + s^k) - f(x^k)$ is compared to the reduction predicted by the quadratic estimate. There are three possibilities:

1. if the prediction is very good, then $x^{k+1} = x^k + s^k$ and the trust region value Δ is increased;

2. if the prediction is moderate, then $x^{k+1} = x^k + s^k$, but the trust region value Δ remains unchanged;
3. if the prediction is poor, then $x^{k+1} = x^k$ and the trust region value Δ is decreased, because the quadratic approximation is apparently not reliable for the current Δ.

Trust region methods need not necessarily use the augmented Lagrangian, but also other objective functions are possible. Furthermore, a trust region method may be combined with one of the previous methods (for example, the ALM), by adding the linearized constraints to (A.15).

A.1.8 Interior point methods (IPM)

An alternative to Lagrangian-based methods is given by barrier methods. In a classical barrier method, terms are added to the objective function such that both the objective function and its first derivative go to infinity when, starting from some feasible point, the solution approaches the boundary of the feasible area. In this way, we can make sure that subsequent iterations will generate points that are feasible, and that they only in the limit may reach (but not cross) the boundary of the feasible region. Methods employing such barrier functions are known as interior point methods. For linear optimization, these methods have been proven to be very successful during the last years, and they are more efficient then the traditional simplex methods, especially for large problems. Interior point methods for general nonlinear programming are being developed nowadays, and they are more and more competitive with traditional algorithms. We will illustrate one of the currently available IPM algorithms for nonlinear optimization, based on [115]. This paper shows also another variant. Other variants exist, *e.g.*, as described in [14].

We start with our standard problem (A.1), where the inequality constraints are turned into equalities by addition of slack variables:

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) + z_i = 0, \quad i = 1, \cdots, m, \\ & h_j(x) = 0, \quad j = 1, \cdots, p, \\ & z_i \geq 0, \quad i = 1, \cdots, m. \end{aligned}$$

In a logarithmic barrier approach, the nonnegativity constraints of the z_i are replaced by logarithmic penalties in the objective:

$$\begin{aligned} \text{minimize} \quad & f(x) - \mu \sum_{i=1}^{m} \ln(z_i) \\ \text{subject to} \quad & g_i(x) + z_i = 0, \quad i = 1, \cdots, m, \\ & h_j(x) = 0, \quad j = 1, \cdots, p. \end{aligned} \tag{A.16}$$

Here, μ is some suitable penalty parameter. The Lagrangian $L_b(x, u, w)$ for (A.16) is given as

$$L_b(x, z, u, w) := f(x) - \mu \sum_{i=1}^{m} \ln(z_i) + \sum_{i=1}^{m} u_i(g_i(x) + z_i) + \sum_{j=1}^{p} w_j h_j(x),$$

and the first order optimality conditions are given by

$$\nabla_x L_b(x,z,u,w) = 0, \tag{A.17}$$

$$\nabla_z L_b(x,z,u,w) = 0, \tag{A.18}$$

$$h_j(x) = 0, \quad j=1,\cdots,p, \tag{A.19}$$

$$g_i(x)+z_i = 0, \quad i=1,\cdots,m. \tag{A.20}$$

Condition (A.18) can be worked out and written as

$$z_i u_i = \mu, \quad i=1,\cdots,m.$$

For $\mu = 0$, the solution of (A.17)-(A.20) is exactly a solution to the KKT conditions of the original problem. So, one might want to use Newton's method for solving the system with $\mu = 0$ directly. It is well-known, however, that Newton's method has good (quadratic) convergence properties only if the initial solution is not 'too far' from the true solution. Interior point methods start with a nonzero μ, for which it is by construction (see below) easier to solve the system of equations. Then μ is gradually reduced to zero.

Interior point methods are based on the assumption that an initial value x^0 is known such that (A.19) and (A.20) are satisfied for nonnegative values z_i^0, $i=1,\cdots,m$, and that values w_j^0, $j=1,\cdots,m$ and $u_i^0 > 0$, $i=1,\cdots,m$ are found such that (A.17) is approximately satisfied[2]. Starting from $k=0$, the kth iteration then proceeds as follows.

1. *Convergence test.* If system (A.17)-(A.20) is satisfied for $\mu = 0$ up to a given precision, then stop.

2. *Search direction.* Let
$$\mu = \sigma^k \frac{(z^k)^T u^k}{m}$$
for some $\sigma^k \in (0,1)$. For this fixed μ, and starting from solution (x^k, z^k, u^k, w^k), a Newton direction $(\delta x, \delta z, \delta u, \delta w)$ for system (A.17)-(A.20) is computed.

3. *Line search.* A step length $\lambda \leq 1$ is computed, to ensure that infeasibility of the system is reduced, that z and u stay positive, and the values of the different products $z_i u_i$ do not diverge,

4. *Update the current point.* $(x,z,u,w)^{k+1} = (x,z,u,w)^k + \lambda(\delta x, \delta z, \delta u, \delta w)$, $k = k+1$, and the algorithm returns to step 1.

This is a very general description and several variants are possible. For linearly-constrained problems, IPM's turn out to be very effective, especially for large-scale problems. For nonlinear problems, some encouraging implementations are available, and there is still a huge amount of research going on by a lot of people in order to find the most efficient methods for different types of problems.

---

[2] This assumption seems rather restricting, but there are some ways to work around it. The current description is only to sketch the ideas of IPMs. More details can be found, *e.g.*, in different papers in [126].

A.2 Tools for nonlinear optimization algorithms

A.2.1 Quasi Newton updates of the (inverse) Hessian

The computation of a search direction in unconstrained optimization, or in a reduced gradient environment, is usually performed using a steepest descent algorithm or by a variant of Newton's method, in which the search direction s^k is solved from the system

$$s^k = -H\nabla f(x^k)$$

where $H = (\nabla^2 f(x^k))^{-1}$, the inverse of the Hessian matrix. This method requires second derivatives, which are not always available. A widely used variant is the use of a *quasi Newton* approach. In this case, the Hessian is not computed explicitly, but its inverse is approximated by using the first derivatives from the previous iterates. The iterative procedure starts as usual at some point x^0 and an initial approximation of H, H_0 is given, usually the identity matrix. Let H_k be the current approximation of the inverse Hessian, let $s^k = -H_k\nabla f(x^k)$ be the search direction, and let $x^{k+1} = x^k + \lambda s^k$ be the next iteration point. Define

$$\begin{aligned} \sigma^k &= \lambda s^k = x^{k+1} - x^k \\ y^k &= \nabla f(x^{k+1}) - \nabla f(x^k). \end{aligned}$$

Assuming that the Hessian is constant (*i.e.,* the objective function is quadratic), it holds that

$$\sigma^k = Hy^k$$

for the exact inverse Hessian matrix H, but generally it is true that

$$\sigma^k \neq H_k y^k.$$

Hence, the vectors σ^k and y^k to update the approximation of the Hessian by addition of a correction term D_k, satisfying the *quasi Newton property:*

$$\sigma^k = (H_k + D_k)y^k.$$

The correction term D_k must be chosen such that information in H_k gathered from previous iterations remains valid. So, if

$$\sigma^i = H_k y^i, \quad i = 1, \cdots, k-1,$$

then the new matrix $H_{k+1} = H_k + D_k$ must satisfy the *hereditary property*

$$\sigma^i = H_{k+1} y^i, \quad i = 1, \cdots, k-1,$$

implying $D_k y^i = 0, \quad i = 1, \cdots, k-1$. A last requirement is that H_{k+1} should be positive definite, so that the next quadratic model has a unique local minimum, and

$$s^k = -H_k \nabla f(x^k)$$

is a downhill direction.

Several rules for computing D_k are known in literature. One of the most popular update rules is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula:

$$D_k = \frac{\tau_k(\sigma^k)(\sigma^k)^T - (\sigma^k)(y^k)^T H_k - H_k(y^k)(\sigma^k)^T}{(\sigma^k)^T(y^k)}$$

where

$$\tau_k = 1 + \frac{(y^k)^T H_k (y^k)}{(\sigma^k)^T(y^k)}.$$

Under some well-defined circumstances, a BFGS update may lead to a matrix H_{k+1} that is not positive semidefinite, but it is easy to work around this situation (see *e.g.*, [45, p.121] for details). In practice, numerical errors also may cause H_{k+1} to be not PSD. This can be avoided by an update strategy in which a Cholesky factorization of the Hessian matrix is updated. For a low-rank quasi Newton update, an update of the Cholesky factorization is relatively fast, and makes sure that the Hessian stays PSD.

The hereditary property with the BFGS formula is only guaranteed if $x^{k+1} = x^k + \lambda s^k$ was obtained by an exact line search. In practice this is seldom a problem. Also for general objective functions, the Hessian is not constant, so information from previous iterations is anyhow of limited use. In some practical algorithms, the (inverse) Hessian approximation is therefore reset to the identity matrix when low progress is made in a number of subsequent iterations, so that the first next iteration is a steepest descent step. This may speed up the convergence of an algorithm.

A.2.2 Line search methods

All discussed algorithms except ALTR require that after the computation of a search direction s^k, a line search is performed in which the following one dimensional optimization problem has to be solved:

$$\underset{\lambda^{\min} \leq \lambda \leq \lambda^{\max}}{\text{minimize}}\ f(x^k + \lambda s^k).$$

In line search algorithms, it is assumed that the function to be minimized is at least *unimodal* in the given range $[\lambda^{\min}, \lambda^{\max}]$, which means that there is some $\hat{\lambda} \in [\lambda^{\min}, \lambda^{\max}]$ such that the function is non-increasing on the interval $[\lambda^{\min}, \hat{\lambda}]$ and non-decreasing on the interval $[\hat{\lambda}, \lambda^{\max}]$. (Note that $\hat{\lambda}$ determines a global minimum on $[\lambda^{\min}, \hat{\lambda}]$; also note that unimodality is a much weaker requirement than convexity).

There are a number of methods that always find the global optimal solution for the line search when the function is unimodal. Among them are golden section search, quadratic interpolation and cubic interpolation. In practical nonlinear optimization, an exact line search is too expensive since it requires too much function evaluations. On the other hand, sufficient decrease of the objective function should be obtained, in order to ensure convergence of the overall optimization algorithm. A practical rule that is often applied is the Goldstein-Armijo rule (see *e.g.*, [45, p.100] or [9, p.281]). This rule, and a possible line search algorithm based on this rule, are discussed in the remainder of this section.

Goldstein-Armijo rule

The Goldstein-Armijo rule defines a range of acceptable step lengths by specifying criteria for the minimal and the maximal allowed step length. Let

$$\theta(\lambda) = f(x^k + \lambda s^k)$$

and hence

$$\theta'(0) = \nabla f(x^k)^T s^k.$$

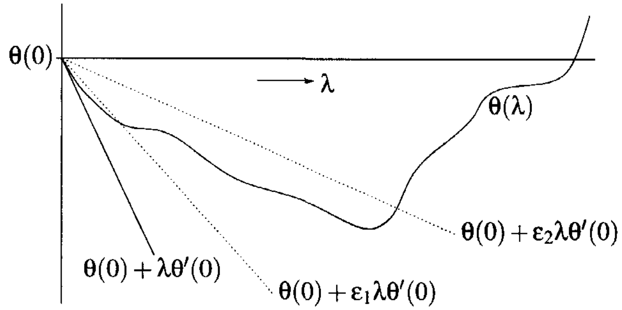
A first order approximation of $\theta(\lambda)$ is given by

$$\theta(0) + \lambda\theta'(0).$$

The Goldstein-Armijo rule relates the actual function value $\theta(\lambda)$ to this approximation and considers a step λ acceptable if

$$\theta(0) + \varepsilon_1\lambda\theta'(0) \leq \theta(\lambda) \leq \theta(0) + \varepsilon_2\lambda\theta'(0) \tag{A.21}$$

for some ε_1 and ε_2 satisfying $1 > \varepsilon_1 \geq \varepsilon_2 > 0$. Graphically this corresponds to the requirement that $\theta(\lambda)$ is in some cone, *cf.* Figure A.1. The Goldstein-Armijo rule may be used in the following step length algorithm.



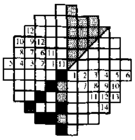
$\theta(0)$
λ
$\theta(\lambda)$
$\theta(0) + \varepsilon_2\lambda\theta'(0)$
$\theta(0) + \lambda\theta'(0)$
$\theta(0) + \varepsilon_1\lambda\theta'(0)$

FIGURE A.1: Illustration of the Goldstein-Armijo criterion. Step length should be such that $\theta(\lambda)$ is in the cone surrounded by the dashed lines.

Goldstein-Armijo based unit step algorithm

This is a fast method in which only function values and no gradients are computed. The method starts with a step length $\bar{\lambda}$. If (A.21) is satisfied, then either $\bar{\lambda}$ is selected as the step size, or a sequence of values $w^i\bar{\lambda}, i = 0, 1, \cdots$ is computed for some factor $w > 1$, to find the largest i that satisfies (A.21). Such a sequence of values is also computed if the left inequality of (A.21) is not satisfied. If on the other hand the right inequality of (A.21) is not satisfied, then a sequence of values $w^{-i}\bar{\lambda}, i = 1, 2, \cdots$ is computed to find the smallest i for which (A.21) is satisfied.

The selection of good values for $\bar{\lambda}$ and w depends on the algorithm being used. One practical setting is $\bar{\lambda} = 1$ and $w = 2$. This setting works well if the step length is expected to be approximately 1, for example when using a Newton search direction on a model that is approximately quadratic.



Appendix B

Some proofs

This appendix contains a collection of mathematical proofs and sketches of proofs for some theorems that were stated in this thesis. These results are needed to justify the validity of the solutions which are obtained with our models.

B.1 Results on eigenvalues and eigenfunctions

All our models contain a space-dependent eigenvalue equation that contains the flux as an eigenfunction and the effective multiplication factor as an eigenvalue. In the text after (2.7) on page 21 it is stressed that it is important to know that the eigenfunction corresponding to the largest eigenvalue is everywhere nonnegative and, up to a constant factor, unique, while no other eigenfunction is nonnegative over the whole space. In Section B.1.1 this is shown for the discrete inverse equation that is actually used in our models. In Section B.1.2 a sketch of a proof, together with pointers to literature, is given for the continuous case.

B.1.1 Discrete inverse equation (2.15)

Theorem B.1 *Consider the discrete eigenvalue equations (2.15)*

$$k_t^{\text{eff}}\phi_{\cdot,t} = GK_t\phi_{\cdot,t}, \quad t = 1,\cdots,T, \tag{B.1}$$

where G is the matrix of discretized Green function values,

$$K_t = \begin{pmatrix} k_{1,t}^{\infty} & & 0 \\ & \ddots & \\ 0 & & k_{I,t}^{\infty} \end{pmatrix},$$

and $\phi_{\cdot,t} = (\phi_{1,t},\cdots,\phi_{I,t})^T$. *Let, according to our normalization constraint (2.16),* $K_t\phi_{\cdot,t}$ *be normalized to a positive constant, which is without loss of generality set to 1:*

$$e^T K_t\phi_{\cdot,t} = 1, \quad t = 1,\cdots,T. \tag{B.2}$$

There exists exactly one nonnegative solution $\phi_{\cdot,t}$ *to (B.1),(B.2). This is the only eigenvector associated with the largest eigenvalue* k_t^{eff} *of (B.1).*

Theorem B.1 follows from the Perron-Frobenius theorem if GK_t is an irreducible matrix. Therefore it suffices to show that GK_t is an irreducible matrix. We start with a definition of irreducibility of a matrix.

Definition B.1 *A nonnegative $n \times n$ matrix A is* irreducible *if for any pair (i,j), $1 \leq i,j \leq n$, there is a positive integer $p = p(i,j) \leq n$ such that $(A^p)_{ij} > 0$.*

Lemma B.1 *The matrix GK_t is irreducible.*

Proof: Since K_t is only a scaling to the matrix G, it is sufficient to show that G is irreducible. Recall the physical meaning of G:

- $G_{i,j}$: the probability that a neutron produced in node j will be absorbed in node i.

The main diagonal of G is strictly positive, so $p(i,i) = 1$, $i = 1,...,n$. For the case $i \neq j$, recall that the probability $G_{i,j}$ is nonzero for two neighboring nodes. Furthermore there is no subset of nodes in the core that is separate from the other nodes, so that each node can be reached by each other node by jumping at most $n-1$ times from one neighbor to another. In other words, given two arbitrary nodes i and j, a chain of nodes $i = i_1, i_2, \cdots, i_p = j$, $p < n$ can be found such that two adjacent nodes in the chain are are neighbors in the core, thus $G_{i_k, i_{k+1}} > 0$ for $k = 1, \cdots, p-1$. This implies that $G^p_{i_1, i_n} > 0$, showing that G is irreducible. □

Theorem B.1 now follows directly from the Perron-Frobenius theorem stated below, that, with a complete proof, can be found in *e.g.*, [7, Th. 1.4.4].

Theorem B.2 (Perron-Frobenius theorem) *Let $A \geq 0$ be an $n \times n$ irreducible matrix. Then*

1. *$Ay = \lambda_0 y$ for some $\lambda_0 > 0$, $y > 0$.*
2. *The eigenvalue λ_0 is simple,* i.e., *its eigenspace is one-dimensional.*
3. *λ_0 is maximal in modulus among all the eigenvalues of A.*
4. *The only nonnegative eigenvectors of A are just the positive scalar multiples of y.*

This theorem shows that there exists exactly one nonnegative, (even strictly positive) solution of the discrete eigenvalue equation in our problem, associated with the largest eigenvalue.

B.1.2 Continuous diffusion equation (2.9)

Theorem B.3 *Consider the diffusion equation (2.9)*

$$-L_f^2 \nabla_r^2 \phi(r) + \phi(r) = \lambda k^\infty(r)\phi(r), \tag{B.3}$$

where $\lambda := \frac{1}{k^{\text{eff}}}$, *with the boundary condition (2.11)*

$$\phi(r) = 0, \text{ where } r \text{ is on the boundary around the core} \tag{B.4}$$

and the normalization from (2.13), that without loss of generality is set to 1:

$$\int_X \phi(r) k^{\infty}(r)\, dr = 1. \tag{B.5}$$

There exists exactly one nonnegative solution ϕ_r *to (B.3),(B.4),(B.5). This is the only eigenfunction associated with the smallest eigenvalue* λ *of (B.3).*

In the one-dimensional case, the diffusion equation is just a special case of the Sturm-Liouville equation. For this equation, many results exist (see *e.g.*, [12, 72]). Among others, we know that all eigenvalues are real and simple, all eigenfunctions are real, and for any two eigenfunctions ϕ_1, ϕ_2, we have

$$\int_X k^{\infty}(r)\phi_1(r)\phi_2(r)\, dr = 0.$$

The most important property for us is given in the Sturm-Oscillation theorem [72]:

Theorem B.4 (Sturm Oscillation Theorem) *There exists an infinitely increasing sequence of eigenvalues* $\lambda_0, \ldots, \lambda_n, \ldots$ *of the boundary-value problem given by (B.3), (B.4), defined over the one-dimensional space* $[a,b]$, *and the eigenfunction corresponding to the eigenvalue* λ_m *has precisely m zero's in the interval* (a,b).

In order to be able to prove Theorem B.3 in the multi-dimensional case, one needs to use the theory of linear operators in Hilbert space. A *Hilbert space* is, roughly speaking, a space of vectors or functions in which certain operators like addition and scalar multiplication are defined, as well as an inner product, and where each Cauchy sequence converges to an element in the space[1]. In our case, the Hilbert space consists of all functions $\phi(r)$ that satisfy the boundary conditions, and the inner product of two functions $\phi_1(r)$ and $\phi_2(r)$ is defined as

$$< \phi_1, \phi_2 >= \int_X \phi_1(r)\phi_2(r)\, dr.$$

(Note that we omit the operand from the functions for clarity of presentation where this is possible.) On this space, one can define operators M, that are positive if the inner product $< M\phi, \phi >$ is positive for any nonzero function ϕ. It is easy to show that

$$M[\phi] := -L_{\text{f}}^2 \nabla^2 \phi + \phi$$

is a positive linear operator in our Hilbert space using Greens' theorem. For any nonzero function ϕ it holds that

$$\begin{aligned} < M[\phi], \phi > &= \int_X \left(-L_{\text{f}}^2 (\nabla_r^2 \phi)\phi + \phi\phi\right) dr \\ &= \int_{\partial X} (-L_{\text{f}}^2 \nabla\phi \cdot \vec{n})\phi\, dr + \int_X L_{\text{f}}^2 (\nabla\phi)^2\, dr + \int_X \phi^2\, dr \\ &> 0, \end{aligned} \tag{B.6}$$

---

[1] For more details and precise definitions of Hilbert spaces and operators one may consult *e.g.*, [48].

where ∂X is the boundary, $\bar{n}$ is the normal vector at the boundary, and the positivity follows since the first term is zero due to the boundary conditions, while the second and third terms are integrals over quadratic functions that are nonzero. Birkhoff [10, 11] mentioned that for positive operators in Hilbert spaces, an earlier theorem of Jentzsch can be extended to show that there exists a unique eigenfunction associated with the smallest eigenvalue. This eigenfunction is strictly positive on the interior of the space where the differential equation is defined, and the associated eigenvalue is also positive.

We still have to justify that none of the other eigenfunctions can be everywhere nonnegative. In the case as is used in our work, the operator M is symmetric, which is due to the fact that the probability for a neutron emitted at x to be absorbed at y is the same as the probability in the opposite direction (for a mathematical definition of symmetric operators see *e.g.*, [48]). The symmetry property implies [48] that all eigenfunctions are real and orthogonal to each other. This implies that the positive eigenfunction associated with the smallest eigenvalue is the only eigenfunction that is everywhere nonnegative.

B.2 Mathematical proof that k^∞ decreases during a cycle

In footnote 3 in Section 5.2.1 on page 93 the following lemma is stated, which is used to sharpen bounds in nonlinear optimization.

Lemma B.2 *From the model equations of the basic reload pattern optimization model, it follows that*

$$k_{i,t}^{\infty,\mathrm{up}} = k^{\mathrm{fresh}}, \quad i = 1,\cdots,I,\ t = 1,\cdots,T.$$

Although this is intuitively clear for physical reasons, we can also deduct a proof from the assignment constraints, the reloading equations and the burnup equations:

$$\sum_{i=1}^{I} V_i x_{i,\ell,m} = 1, \qquad \ell = 1,\cdots,L,\ m = 1,\cdots,M \tag{B.7}$$

$$\sum_{\ell=1}^{L}\sum_{m=1}^{M} x_{i,\ell,m} = 1, \qquad i = 1,\cdots,I \tag{B.8}$$

$$\begin{aligned} k_{i,1}^{\infty} &= \sum_{m=1}^{M} x_{i,1,m} k^{\mathrm{fresh}} \\ &+ \sum_{\ell=2}^{L}\sum_{m=1}^{M} x_{i,\ell,m} \sum_{j=1}^{I} V_j x_{j,\ell-1,m} k_{j,T}^{\infty}, \qquad i = 1,\cdots,I \end{aligned} \tag{B.9}$$

$$k_{i,t+1}^{\infty} = k_{i,t}^{\infty} - \alpha \Delta_t k_{i,t}^{\infty} R_{i,t}, \qquad i = 1,\cdots,I,\ t = 1,\cdots,T-1. \tag{B.10}$$

In the proof we will use the notation

$$k_{\cdot,t}^{\infty} := (k_{1,t}^{\infty},\cdots,k_{I,t}^{\infty})^T.$$

For ease of notation we further introduce

$$A_{ij} := \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} V_j x_{j,\ell-1,m}, \quad \beta_i := \sum_{m=1}^{M} x_{i,1,m} \tag{B.11}$$

Using this notation, the reloading equations (B.9) are written in matrix form as

$$k^\infty_{\cdot,1} = Ak^\infty_{\cdot,T} + \beta k^{\text{fresh}}. \tag{B.12}$$

Before starting the actual proof of Lemma B.2 we state and proof another lemma.

Lemma B.3 *Let A be given as defined in (B.11), then for any vector z,*

$$(I-A)z \le 0 \Rightarrow z \le 0. \tag{B.13}$$

Proof of Lemma B.3: Suppose that $z_i > 0$ for one or more i. We will show that then also $[(I-A)z]_j > 0$ for some j. Let

$$p = \underset{s=1,\cdots,I}{\operatorname{argmax}}\, z_s. \tag{B.14}$$

Using the definition of A and the assignment constraints (B.7) and (B.8), it holds that

$$\begin{aligned} [(I-A)z]_p &= z_p - \sum_{\ell=2}^{L}\sum_{m=1}^{M} x_{p,\ell,m} \sum_{s=1}^{I} V_s x_{s,\ell-1,m} z_s \\ &\ge z_p - \sum_{\ell=2}^{L}\sum_{m=1}^{M} x_{p,\ell,m} \sum_{s=1}^{I} V_s x_{s,\ell-1,m} z_p && \text{(since } z_p \ge z_s\text{)} \\ &= z_p - \sum_{\ell=2}^{L}\sum_{m=1}^{M} x_{p,\ell,m} z_p && \text{(using (B.7))} \\ &\ge 0 && \text{(using (B.8)).} \end{aligned} \tag{B.15}$$

We now show that a p exists such that the inequality is strict. Suppose that equality in (B.15) is obtained for some i that satisfies (B.14), *i.e.*,

$$z_i = \max_s z_s > 0.$$

Let us consider the consequences.

1. $x_{i,1,m} = 0$ for all m due to the last inequality of (B.15), hence at least one pair (ℓ, m), $\ell > 1$ exists with $x_{i,\ell,m} > 0$. Let $\tilde{\ell}$ be the smallest ℓ for which such a pair exists.

2. Given this pair $(\tilde{\ell}, m)$ there exists a r such that $x_{r,\tilde{\ell}-1,m} > 0$. Furthermore, $r \ne i$ since $\tilde{\ell}$ was chosen as the smallest ℓ for which $x_{i,\ell,m} > 0$.

3. Since $x_{i,\tilde{\ell},m} > 0$ and $x_{r,\tilde{\ell}-1,m} > 0$ it holds that $z_r = z_i$; otherwise the first inequality of (B.15) would have been strict for i. This implies that r is another solution of (B.14), and (B.15) also holds for $p = r$, but we know that $x_{r,\ell,m} > 0$ for $\ell = \tilde{\ell} - 1$. If (B.15) still holds with equality for $i = r$, then we can repeat our reasoning from step 1 above, but ultimately

we will find an index j such that still $z_j = z_i > 0$, but either the first inequality of (B.15) becomes strict, or otherwise $x_{j,1,m} > 0$ for some m, hence the last inequality of (B.15) becomes strict. In any case we find that

$$z_i > 0 \rightarrow [(I-A)z]_j > 0 \text{ for some } j$$

which proves (B.13).[2] □

Now we are ready for the **Proof of Lemma B.2:** From the burnup equations (B.10) and non-negativity of $R_{i,t}$ it follows that $k^{\infty}_{i,T} \leq k^{\infty}_{i,1}$, hence the reloading equation (B.12) can be rewritten as

$$(I-A)k^{\infty}_{\cdot,1} \leq \beta k^{\text{fresh}}. \tag{B.16}$$

From the assignment constraints (B.7) and (B.8) it follows that

$$\beta_i + (Ae)_i = \sum_{m=1}^{M} x_{i,1,m} + \sum_{\ell=2}^{L} \sum_{m=1}^{M} x_{i,\ell,m} \cdot 1 = 1,$$

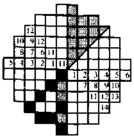
so that $\beta = (I-A)e$, and (B.16) is written as

$$(I-A)(k^{\infty}_{\cdot,1} - ek^{\text{fresh}}) \leq 0. \tag{B.17}$$

Application of Lemma B.3 with $z := k^{\infty}_{\cdot,1} - ek^{\text{fresh}}$ proves Lemma B.2. □

---

[2] An alternative proof of (B.13) was possible using the theory of Markov chains. It can be shown that A is an (incomplete) transition matrix without closed subsets, for which (B.13) is a well-known property (*e.g.*, [50]).



Bibliography

[1] E. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization.* Wiley & Sons, Chichester, 1997.

[2] D.H. Ahn and S.H. Levine. Direct placement of fuel assemblies using the gradient projection method. *Transactions of the American Nuclear Society*, 46:123–125, 1984.

[3] V.V. Amel'kin. *Differential Equations in Applications.* Mir Publishers, Moscow, 1990.

[4] I.P. Androulakis, C.D. Maranas, and C.A. Floudas. αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7:337–363, 1995.

[5] J.K. Axmann. Parallel adaptive evolutionary algorithms for pressurized water reactor reload pattern optimizations. *Nuclear Technology*, 119:276–292, 1997.

[6] J.W. Bandler, R.M. Biernacki, S.Hua Chen, R.H. Hemmers, and K. Madsen. Electromagnetic optimization exploiting aggressive space mapping. *IEEE Transactions on Microwave Theory and Techniques*, 43:2874–2882, 1995.

[7] R.B. Bapat and T.E.S. Raghavan. *Nonnegative Matrices and Applications.* Cambridge University Press, Cambridge, 1997.

[8] C. Barnhart, J. Desrosiers, and M.M. Solomon (eds.). Focused issue on air transportation. *Transportation Science*, 32(3):205–301, 1998.

[9] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty. *Nonlinear Programming. Theory and Algorithms.* Wiley & Sons, New York, 1993.

[10] G. Birkhoff. Extension of Jentzsch's theorem. *Transactions of the American Mathematical Society*, 85:219–227, 1957.

[11] G. Birkhoff. Reactor criticality in neutron transport theory. *Rendiconti di Matematica e delle sue Applicazioni*, 22:1–25, 1963.

[12] W.E. Boyce and R.C. DiPrima. *Elementary Differential Equations and Boundary Value Problems.* Wiley & Sons, 1986.

[13] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide, Release 2.25.* The Scientific Press, 1992.

[14] R.H. Byrd, M.E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. July 1997.

[15] J.N. Carter. Genetic algorithms for incore fuel management and other recent developments in optimisation. *Advances in Nuclear Science and Technology*, 25:113–154, 1997.

[16] H. Casanova, J.J. Dongarra, and K. Moore. Network-enabled solvers and the netsolve project. *SIAM News*, 31(1):1,8, 1998.

[17] Y. Censor and S.A. Zenios. *Parallel Optimization, Theory, Algorithms and Applications.* Oxford University Press, New York, Oxford, 1997.

[18] R.G. Cochran and N. Tsoulfanidis. *The Nuclear Fuel Cycle: Analysis and Management.* American Nuclear Society, La Grange Park, Illinois, USA, 1992.

[19] A.R. Conn, N.I.M. Gould, and Ph. L. Toint. *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A).* Springer-Verlag, Berlin, 1992.

[20] A.R. Conn, K. Scheinberg, and Ph.L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In M.D. Buhmann and A. Iserles, editors, *Approximation Theory and Optimization*, pages 83–108. Cambridge University Press, 1997.

[21] A.R. Conn, K. Scheinberg, and Ph.L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79:397–414, 1997.

[22] Cute: Constrained and unconstrained testing environment. `http://www.cse.clrc.ac.uk/Activity/CUTE`.

[23] G.B. Dantzig. Programming in a linear structure. *Econometrica*, 17:73–74, 1948. Report of the Sept. 9, 1948 meeting in Madison.

[24] G.B. Dantzig and M.N. Thapa. *Linear Programming 1: Introduction.* Springer, New York, 1997.

[25] M.D. DeChaine and M.A. Feltus. Fuel management optimization using genetic algorithms and expert knowledge. *Nuclear Science and Engineering*, 124:188–196, 1996.

[26] M. Dell'Amico, F. Maffioli, and S. Martello. *Annotated Bibliographies in Combinatorial Optimization.* Wiley & Sons, Chichester, 1997.

[27] A.S. Drud. A large-scale GRG code. *ORSA Journal on Computing*, 6(2):207–216, 1992.

[28] J.J. Duderstadt and L.J. Hamilton. *Nuclear Reactor Analysis.* Wiley & Sons, 1976.

[29] M. Dumas and D. Robeau. Fuel management optimization for a PWR. In *Proc. Int. Topical Meeting on Advances in Math. Methods for the Solution of Nuclear Engineering Problems*, pages V.2 357–367. München, 1981.

[30] M.A. Duran and I.E. Grossmann. An outer approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307, 1986.

[31] J. Beasley (ed.). *Advances in Linear and Integer Programming*. Oxford University Press, Oxford, 1996.

[32] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. Classics in Applied Mathematics. SIAM, Philadelphia, 1990. Formerly published by Wiley, New York, 1968.

[33] R. Fletcher and S. Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[34] C.A. Floudas. *Nonlinear and Mixed-Integer Optimization; Fundamentals and Applications*. Topics in Chemical Engineering. Oxford University Press, New York, 1995.

[35] E. Galligani, V. Ruggiero, and L. Zanni. Parallel solution of large scale quadratic programs. In R. De Leone *et al.* , editor, *High Performance Algorithms and Software in Nonlinear Optimization*, pages 189–205. Kluwer Academic Publishers, Boston, 1998.

[36] A. Galperin and Y. Kimhy. Application of knowledge-based methods to in-core fuel management. *Nuclear Science and Engineering*, 109:103–110, 1991.

[37] GAMS Development Corporation, Washington, DC. *GAMS - The Solver Manuals*.

[38] R. van Geemert. Evaluation and optimization of fuel assembly shuffling schemes for a nuclear reactor core. Technical Report IRI-131-95-016, Delft University of Technology, 1995.

[39] R. van Geemert. *Nuclear Reactor Reload Pattern Optimization by Application of Heuristic Search and Perturbation Theoretical Methods*. PhD thesis, Delft University of Technology, 1999.

[40] R. van Geemert and J.E. Hoogenboom. A two-dimensional analytical nodal core model for use in a reload pattern optimization procedure. In *Proceedings of the 9th Topical Meeting, Moscow, MEPhI, h/c "Volga"*, September 4–8, 1994.

[41] R. van Geemert, J.E. Hoogenboom, and A.J. Quist. In-core fuel management optimization by incorporation of a perturbation theoretical approach in a heuristic search procedure. In *Proceedings International ANS Conference on the Physics of Nuclear Reactors (PHYSOR'98), Long Island (New York)*, pages V1.91–101, 1998.

[42] R. van Geemert, A.J. Quist, and J.E. Hoogenboom. Reload pattern optimization by application of multiple cyclic interchange algorithms. In *Proceedings of PHYSOR96*, pages I38–I47. Mito, Japan, 1996.

[43] R. van Geemert, A.J. Quist, and J.E. Hoogenboom. Application of depletion perturbation theory and sensitivity analysis for minimizing the required feed enrichment for an equilibrium cycle. In *Proceedings of the Topical Meeting Advances in Nuclear Fuel Management II, Volume 2*, pages 15-1–15-9. American Nuclear Society, 1997.

[44] R. van Geemert, A.J. Quist, J.E. Hoogenboom, and H.P.M. Gibcus. Research reactor in-core fuel management optimization by application of multiple cyclic interchange algorithms. *Nuclear Engineering and Design*, 186:369–377, 1998.

[45] P.E. Gill, W. Murray, and M.H. Wright. *Practical Optimization.* Academic Press, 1981.

[46] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549, 1986.

[47] F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1:190, 1989.

[48] I. Gohberg and S. Goldberg. *Basic Operator Theory.* Birkhäuser, 1981.

[49] R.E. Griffith and R.A. Stewart. A nonlinear programming technique for the optimization of continuous processing systems. *Management Science*, 7:379–392, 1961.

[50] G.R. Grimmett and D.R. Stirzaker. *Probability and Random Processes.* Oxford University Press, Oxford, 1990.

[51] A. Hertz and D. Kobler. A framework for the description of population based methods. Technical Report RR ORWP 98/04, Dept. of Mathematics, EPF-Lausanne, Switzerland, 1998.

[52] G.H. Hobson and P.J. Turinsky. Automatic determination of pressurized water reactor core loading patterns that maximize beginning-of-cycle reactivity within power-peaking and burnup constraints. *Nuclear Technology*, 74:5–13, 1986.

[53] K. Holmberg. On the convergence of the cross decomposition. *Mathematical Programming*, 47:269, 1990.

[54] J.E. Hoogenboom and H. van Dam. Reactor physics (in dutch). Lecture Notes Delft University of Technology, 1994.

[55] R. Horst and P.M. Pardalos. *Handbook of Global Optimization*, volume 2 of *Nonconvex Optimization and its Applications.* Kluwer Academic Publishers, Dordrecht, 1995.

[56] T. Hoshino. In-core fuel management optimization by heuristic learning technique. *Nuclear Science and Engineering*, 49:59–71, 1972.

[57] T. Hoshino, M. Takahashi, and Y. Fujii. Optimization of in-core fuel management, cycle period and power scheduling of nuclear power plants by large scale nonlinear programming. In *Proc. Conf. on Comp. Methods in Nuclear Engineering*, pages V1. III-115–III-134. Charlestone, South Carolina, April 15–17, 1975.

[58] ILOG Inc., Incline Village. *Using the CPLEX Callable Library.* Version 6.0.

[59] A.J. de Jong. Reloading pattern design for batch refuelled nuclear reactors. Technical Report IRI 131-95-010, Delft University of Technology, 1995.

[60] H.G. Kim, S.H. Chang, and B.H. Lee. Optimal fuel loading pattern design using an artificial neural network and a fuzzy rule-based system. *Nuclear Science and Engineering*, 115:152–163, 1993.

[61] H.G. Kim, S.H. Chang, and B.H. Lee. Pressurized water reactor core parameter prediction using an artificial neural network. *Nuclear Science and Engineering*, 113:70–76, 1993.

[62] T.K. Kim and C.H. Kim. Determination of optimized PWR fuel loading pattern by mixed integer programming. In *Proceedings of PHYSOR96*, pages I76–I85. Mito, Japan, 1996.

[63] T.K. Kim and C.H. Kim. Mixed integer programming for pressurized water reactor fuel-loading-pattern optimization. *Nuclear Science and Engineering*, 127:346–357, 1997.

[64] Y.J. Kim, T.J. Downar, and A. Sesonske. Optimization of core reload design for low-leakage fuel management in pressurized water reactors. *Nuclear Science and Engineering*, 96:85–101, 1987.

[65] E. de Klerk, T. Illés, A.J. de Jong, C. Roos, T. Terlaky, J. Valkó, and J.E. Hoogenboom. Optimization of nuclear reactor reloading patterns. *Annals of Operations Research*, 69:65–84, 1997.

[66] D.J. Kropaczek and P.J. Turinsky. In-core nuclear fuel management optimization for pressurized water reactors utilizing simulated annealing. *Nuclear Technology*, 95:9–32, 1991.

[67] T. Kubokawa and R. Kiyose. Optimization of in-core fuel management by integer linear programming. In *Proc. Conf. on Comp. Methods in Nuclear Engineering*, pages V. III–135–III–149. Charlestone, South Carolina, April 15–17, 1975.

[68] H.W. Kuhn. Nonlinear programming: A historical note. In Lenstra et al. [71], pages 82–96.

[69] J.C. Lagarias, J.A. Reeds, M.H. Wright, and P.E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal of Optimization*, 9(1):112–147, 1998.

[70] J.-L. Lagrange. *Théorie des fonctions analytiques*. 1813. Reprinted as: J.-A. Serret (ed.) (1881). *Oeuvres de Lagrange, Tome IX,* Gauthier-Villars, Paris.

[71] J.K. Lenstra, A.H.G. Rinnooy Kan, and A. Schrijver, editors. *History of Mathematical Programming: A Collection of Personal Reminiscences*. Elsevier Science Publishers, Amsterdam, 1991.

[72] B.M. Levitan and I.S. Sargsjan. *Sturm-Liouville and Dirac Operators*. Kluwer Academic Publishers, 1991.

[73] S. Leyffer. Integrating SQP and Branch-and-Bound for mixed integer nonlinear programming. Technical Report NA/182, Dundee University Numerical Analysis Report, 1998.

[74] S. Leyffer. Personal communication, Fall 1997. Dundee University.

[75] C. Lin, J-I. Yang, K-J. Lin, and Z-D. Wang. Pressurized water reactor loading pattern design using the simple tabu search. *Nuclear Science and Engineering*, 129:61–71, 1998.

[76] F.A. Lootsma. Parallel non-linear optimization. In H.W. Tyrer, editor, *Advances in Distributed and Parallel Processing Volume Two: Applications in Optimization, Fluid Dynamics, and VLSI*, chapter 2, pages 5–33. Ablex Publishing Corporation, Norwood, NJ, 1994.

[77] Y.P. Mahlers. Core loading pattern optimization for pressurized water reactors. *Annals of Nuclear Energy*, 21:223–227, 1994.

[78] Y.P. Mahlers. Core loading pattern optimization for research reactors. *Annals of Nuclear Energy*, 24(7):509–514, 1997.

[79] G.I. Maldonado and P.J. Turinsky. Application of nonlinear nodal diffusion generalized perturbation theory to nuclear fuel reload optimization. *Nuclear Technology*, 110:198–219, 1995.

[80] G.I. Maldonado, P.J. Turinsky, and D.J. Kropaczek. Employing nodal generalized perturbation theory for the minimization of feed enrichment during pressurized water reactor in-core fuel management optimization. *Nuclear Science and Engineering*, 121:312–325, 1995.

[81] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I - convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[82] J.O. Mingle. In-core fuel management via perturbation theory. *Nuclear Technology*, 27:248–257, 1975.

[83] H. Motoda, J. Herczeg, and A. Sesonske. Optimization of refueling schedule for light-water reactors. *Nuclear Technology*, 25:477–496, 1975.

[84] T.S. Motzkin. Signs of minors. In O. Shisha, editor, *Inequalities*, pages 225–240. Academic Press, 1967.

[85] K.G. Murty. *Linear Programming*. Wiley & Sons, New York, 1983.

[86] K.G. Murty. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987.

[87] B.N. Naft and A. Sesonske. Pressurized water reactor optimal fuel management. *Nuclear Technology*, 14:123–132, 1972.

[88] S.G. Nash and A. Sofer. *Linear and Nonlinear Programming*. McGraw-Hill, New York, 1996.

[89] J.L. Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, New York, 1987.

[90] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley & Sons, New York, 1988.

[91] NEOS: Network enabled optimization software server. `http://www-neos.mcs.anl.gov/neos/`.

[92] Y. Nesterov and A.S. Nemirovskii. *Interior Point Polynomial Algorithms in Convex Programming*. SIAM Studies in Applied Mathematics, Vol. 13. SIAM, Philadelphia, 1994.

[93] NetSolve:. `http://www.cs.utk.edu/netsolve/`.

[94] E. Nissan and A. Galperin. Refueling in nuclear engineering: the FUELCON project. *Computers in Industry*, 37:43–54, 1998.

[95] A. Osyczka. *Multicriterion Optimization in Engineering*. Wiley, New York, 1984.

[96] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization*. Prentice-Hall, Englewood Cliffs, 1982.

[97] G.T. Parks. An intelligent stochastic optimization routine for in-core fuel cycle design. *Transactions of the American Nuclear Society*, 57:259–260, 1988.

[98] G.T. Parks. Advances in optimization and their applicability to problems in the field of nuclear science and technology. *Advances in Nuclear Science and Technology*, 21:195–253, 1990.

[99] G.T. Parks. An intelligent stochastic optimization routine for nuclear fuel cycle design. *Nuclear Technology*, 89:233–246, 1990.

[100] G.T. Parks. Multi-objective pressurized water reactor reload core design by nondominated genetic algorithm search. *Nuclear Science and Engineering*, 124:178–187, 1996.

[101] J. Pintér. *LGO - A Model Development System for Continuous Global Optimization. User's Guide.* Pintér Consulting Services, Halifax, NS, 3rd revised edition.

[102] J.D. Pintér. *Global Optimization in Action*, volume 6 of *Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht, 1996.

[103] S. Poljak, F. Rendl, and H. Wolkowicz. A recipe for semidefinite relaxation for 0,1-quadratic programming. *Journal of Global Optimization*, 7:51–73, 1995.

[104] P.W. Poon and G.T. Parks. Application of genetic algorithms to in-core fuel management optimization. In *Proceedings Joint International Conference on Mathematics and Supercomputing in Nuclear Applications*, pages Vol. 1, 777. Karlsruhe, 1993.

[105] M.J.D. Powell. Direct search algorithms for optimization calculations. In A. Iserles, editor, *Acta Numerica, Vol. 7*, pages 287–336. Cambridge University Press, 1998.

[106] A.J. Quist, R. van Geemert, J.E. Hoogenboom, T. Illés, E. de Klerk, C. Roos, and T. Terlaky. Application of nonlinear optimization to reactor core fuel reloading. *Annals of Nuclear Energy*, 26(5):423–448, 1998.

[107] A.J. Quist, E. de Klerk, C. Roos, and T. Terlaky. Copositive relaxation for general quadratic programming. *Optimization Methods and Software*, 9:185–208, 1998.

[108] M.V. Ramana. *An Algorithmic Analysis of Multiquadratic and Semidefinite Programming Problems*. PhD thesis, The Johns Hopkins University, Baltimore, 1993.

[109] M.V. Ramana and P.M. Pardalos. Semidefinite programming. In Terlaky [126], pages 369–398.

[110] C. Roos, T. Terlaky, and J.-Ph Vial. *Theory and Algorithms for Linear Optimization; an Interior Point Approach*. Wiley & Sons, Chichester, 1997.

[111] H.S. Ryoo and N.V. Sahinidis. A Branch-and-Reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–139, 1996.

[112] N.V. Sahinidis. *BARON: A General Purpose Global Optimization Software Package*. University of Illinois, 1995.

[113] T.O. Sauar. Application of linear programming to in-core fuel management optimization in light water reactors. *Nuclear Science and Engineering*, 46:274–283, 1971.

[114] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley & Sons, Chichester, 1986.

[115] D.F. Shanno, M.G. Breitfeld, and E.M. Simantiraki. Implementing barrier methods for nonlinear programming. In Terlaky [126], pages 399–414.

[116] N.Z. Shor. Quadratic optimization problems. *Soviet Journal of Computer and System Sciences*, 25(6):1–11, 1987.

[117] T. Šmuc, D. Pevec, and B. Petrović. Annealing strategies for loading pattern optimization. *Annals of Nuclear Energy*, 21:325–336, 1994.

[118] W.M. Stacey, Jr. *Space-Time Nuclear Reactor Kinetics*. Academic Press, 1969.

[119] R.E. Steuer. *Multiple Criteria Optimization Theory, Computation, and Application*. Wiley, New York, 1986.

[120] J.G. Stevens, K.S. Smith, K.R. Rempe, and T.J. Downar. Optimization of pressurized water reactor shuffling by simulated annealing with heuristics. *Nuclear Science and Engineering*, 121:67–88, 1995.

[121] R.B. Stout and A.H. Robinson. Determination of optimum fuel loadings in pressurized water reactors using dynamic programming. *Nuclear Technology*, 20:86–102, 1973.

[122] J.S. Suh and S.H. Levine. Optimized automatic reload program for pressurized water reactors using simple direct optimization techniques. *Nuclear Science and Engineering*, 105:371–382, 1990.

[123] É.D. Taillard, L.M. Gambardella, M. Gendreau, and J.-Y. Potvin. Adaptive memory programming: A unified view of metaheuristics. Technical Report IDSIA-19-98, IDSIA, Lugano (Switzerland), 1998.

[124] M. Tawarmalani, S. Ahmed, and N.V. Sahinidis. Convex extensions of l.s.c. functions and reformulations of rational functions of 0-1 variables. Submitted to Mathematical Programming, tawarmal@alexander.scs.uiuc.edu, 1999.

[125] A.B. Tayler. *Mathematical Models in Applied Mechanics*. Clarendon Press, Oxford, 1986.

[126] T. Terlaky, editor. *Interior Point Methods of Mathematical Programming*. Kluwer Academic Publishers, Dordrecht, 1996.

[127] W.B. Terney and E.A. Williamson, Jr. The design of reload cores using optimal control theory. *Nuclear Science and Engineering*, 82:260–288, 1982.

[128] Ph. L. Toint. Private communication, April-May, 1998. FUNDP.

[129] V. Torczon. *Multi-directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Rice University, Houston, 1989.

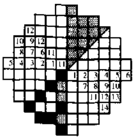
[130] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1996.

[131] S. Wei and C. Pingdong. A new approach for low-leakage reload core multi-cycle optimization design. In *Proceedings of PHYSOR96*, pages I86–I95. Mito, Japan, 1996.

[132] A. Yamamoto. Comparison between equilibrium cycle and successive multicycle optimization methods for in-core fuel management of pressurized water reactors. In *Proceedings of the Joint International Conference on Mathematical Methods and Supercomputing for Nuclear Applications*, pages 769–781, Saratoga Springs, New York, 1997. American Nuclear Society, Inc.

[133] P.L. Yu and M. Zeleny. The set of all nondominated solutions in linear cases and a multicriteria simplex method. *Journal of Mathematical Analysis and Applications*, 49:430–468, 1975.

[134] N. Zavaljevski. A model for fuel shuffling and burnable absorbers optimization in low leakage PWR's. *Annals of Nuclear Energy*, 17(4):217–220, 1990.



List of symbols

Symbols related to the physical model

α	(in basic model) Scaling constant, 36
α^{p}	(in BP model) Scaling constant, 38
χ	Energy distribution function of fission neutrons, 19
D	Diffusion coefficient, 19
δ^{PP}	Measure of power peak constraint violation in the cut algorithm, 111
Δ_t	Length of the time interval starting at time step t, 25
f^{lim}	Maximum allowed ratio between maximum and average local power peak, 35
G	Green function matrix, 23
k^{eff}	Effective multiplication factor, 20
k^{∞}	Infinite multiplication factor, 22
$\overline{k}^{\infty}$	Artificial infinite multiplication factor where BP is neglected, 33
k^{fresh}	Infinite multiplication factor of a fresh bundle, 25
K_t	$I \times I$ diagonal matrix with entries $k^{\infty}_{i,t}$ on the main diagonal, 187
L_{f}	Fast diffusion length, 22
μ_{f}	Fission energy, 22
N	Particle density (which particle should be clear from the context), 16
ν	Average number of neutrons produced per fission, 19
P_{c}	Constant power level, 22
ϕ	Neutron flux, 18
R	Neutron removal rate, 36
Σ	Macroscopic cross section, 17
σ	Microscopic cross section, 16
$\Sigma^{\text{p,fresh}}_{\text{a},m}$	BP cross section in the fresh bundle of trajectory m, 33
V_i	Volume of a node i, 34
v	Neutron velocity, 18
$x_{i,\ell,m}$	Binary variable indicating whether node i contains a bundle of age ℓ from trajectory m, 32

Sub- and superscripts on cross sections

a	absorption, 17
c	capture, 18
f	fission, 17
p	applied to Burnable Poisons only, 26
s	scattering, 17

Model size constants

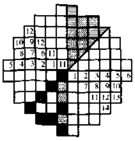
I	Number of nodes in an octant core, counting each individual diagonal node as one, 31
L	Number of age groups in the core, 31
M	Number of trajectories in an octant core, 32
T	Number of time steps in the discretization (*i.e.*, the number of time intervals plus one), 24

Assumptions on the names of indexing subscripts

(g)	Index on energy group
i,j	Indices on node numbers
ℓ	Index on age group
m	Index on trajectory number
t	Index on time step
q	Index on nuclide type
BoC	Time index $t = 1$
EoC	Time index $t = T$

Mathematical symbols

$\boldsymbol{B}$	Cone of completely positive matrices, 159
$\boldsymbol{N}$	Cone of nonnegative matrices, 159
$\boldsymbol{S}$	Cone of positive semidefinite matrices, 158
$\delta(r)$	Dirac δ function, 23
H	Hessian (second derivative) matrix
$L(x,u,w)$	Lagrange function, 59
∇_x	Gradient with respect to x
$\mathrm{Tr}(X)$	Trace of a matrix X (sum of its diagonal elements), 157



List of abbreviations

αBB	a software package for global optimization, 118, 155
AI	Artificial Intelligence, 49
ALM	Augmented Lagrangian method, 67, 178
ALTR	Augmented Lagrangian Trust Region method, 67, 181
AMPL	a modeling language for optimization problems, 153
AP	Augmented Penalty, 75
BARON	a software package for global optimization, 118, 156
BFGS-formula	Broyden-Fletcher-Goldfarb-Shanno formula, 185
BoC	Begin of Cycle, 13
BP	Burnable Poison, 26
CONOPT	a software package for NLP, 117
CPLEX	a software package for MIP, 117
CPU	Central processing unit, 127
CR	our algorithm using CONOPT and a rounding algorithm, 145
CRP	our algorithm using CONOPT, rounding, and PI, 145
CRPR	our algorithm using CONOPT, rounding, and modified PI, 145
CUTE	Library of NLP test problems, 155
DC	our algorithm using DICOPT, 145
DICOPT	a software package for MINLP, 118
DLL	Dynamic Link Library, 117
DONLP2	a software package for NLP, 153
EA	Evolutionary Algorithms, 46
EoC	End of Cycle, 25
ER	Equality Relaxation, 75
FILTER	a software package for NLP, 154
GA	Genetic Algorithms, 42
GAMS	a modeling language for optimization problems, 117
GBD	Generalized Benders Decomposition, 78
GCD	Generalized Cross Decomposition, 78
GOA	Generalized Outer Approximation, 76
GPT	Generalized Perturbation Theory, 52
GQP	General Quadratic Programming, 157
GRG	Generalized Reduced Gradient method, 68, 180
GUI	Graphical User Interface, 117

IPM	Interior Point method, 69, 182
KKT-conditions	Karush-Kuhn-Tucker conditions, 59, 175
LANCELOT	a software package for NLP, 117, 154
LGO	a software package for global optimization, 118, 156
LNZ	Linear nonzero's, 131
LOQO	a software package for NLP, 153
LP	Linear Programming, 55
LWR	Light Water Reactor, 53
MI	Multiple Interchange, 43
MILP	Mixed-integer Linear Programming, 75
MIMD	Multiple Instruction Multiple Data, 160
MINLP	Mixed-integer Nonlinear Programming, 55
MINOS5	a software package for NLP, 117
MIP	Mixed-Integer Programming, commonly used in the sense of MILP
MP	Mathematical Programming, 55
NEOS-server	a web server for optimization, 117
NLNZ	Nonlinear nonzero's, 131
NLP	Nonlinear Programming, 57
OA	Outer Approximation, 74
PD	Positive definite, 61
PI	Pairwise Interchange, 43
PIR	our modified PI algorithm, 145
PLM	Projected Lagrangian Method, 177
PMA	Population Mutation Annealing, 48
PSD	Positive semidefinite, 61
PT	Perturbation Theory, 52
PWR	Pressurized Water Reactor, 53
QP	Quadratic (convex) Programming, 153
RGM	Reduced Gradient Method, 66, 178
RHS	Righthand side (of a constraint)
SA	Simulated Annealing, 45
SDO	Semidefinite optimization, 156
SIF	Standard input format for NLP models, 117
SIMD	Single Instruction Multiple Data, 160
SISD	Single Instruction Single Data, 160
SLP	Sequential Linear Programming, 65, 176
SNOPT	a software package for NLP, 154
SOS	Special Ordered Set, 141
SQP	Sequential Quadratic Programming, 67, 177
TS	Tabu Search, 47
ZOOM	a software package for MIP, 118

Index

artificial intelligence, 49
atomic number, 15
augmented lagrangian method, 67, 178
augmented lagrangian trust region method, 67, 181

barrier function, 69
basic solution, 179
basic variables, 179
basis, 179
binding energy, 15
black box optimization, 58, 79
bounds cooling, 46
branch and reduce, 80
Branch-and-Bound, 51, 72, 105
burnable absorbers, *see* burnable poison
burnable poison, 26, 147

caption, 16
cauchy point, 181
completely positive matrix, 159
control rod, 3
control theory, 51
convex envelope, 81
convex hull, 77
convexity, 56
copositive optimization, 158
cross section
    macroscopic, 17
    microscopic, 16
cross-over, 47
cuts, 76, 108–117, 146

derivative based methods, 42
direct search methods, 42
dynamic programming, 44

effective multiplication factor, 20
engineering constraints, 52, 97
equilibrium cycle condition, 32
evolutionary algorithms, 46

fast diffusion length, 22
fast group, 20
first order conditions, 59
fission, 16
flux, *see* neutron flux
fuel breeding, 29
fuel bundle, 4
fuel element, *see* fuel bundle
fuel management, 1

generalized benders decomposition, 78
generalized cross decomposition, 78
generalized reduced gradient method, 68, 180
global optimal solution, 56
global optimization, 79, 155
global to local search, 48
Goldstein-Armijo rule, 186
Green function, 23

Hessian, 61, 184
Hilbert space, 189
hybrid algorithm, 49

infinite multiplication factor, 22
interior point methods, 69, 182
irreducible matrix, 188
isotope, 15

KKT-conditions, 59, 175

Lagrange
    function, 59, 175
    multipliers, 59, 175
line search, 62, 183, 185

Lipschitz continuity, 79
loading pattern, 13, 31
local optimal solution, 56
local search methods, 42

Markov chain, 192
mass number, 15
mathematical optimization, 6
Monte Carlo method, 49
multigroup diffusion equations, 20
multiple interchange, 43
mutation, 47

neural networks, 49
neutron, 15
neutron density, 17
neutron flux, 18
neutron transport equation, 18
Newton
    direction, 63
    method, 63
nucleus, 15
nuclide, 15
    density, 16

outer approximation, 74, 104

pairwise interchange, 43, 107, 142
parallelization, 159–165
Perron-Frobenius theorem, 188
perturbation theory, 52
population mutation annealing, 48
positive semidefinite matrix, 61, 158
power peaking factor, 35
projected lagrangian method, 177
proton, 15

quasi Newton, 184

reduced gradient, 179
reduced gradient method, 66, 178
reload pattern, *see* loading pattern
rounding, 103
rounding heuristics, 71

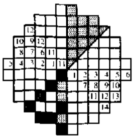
scattering, 16
second order condition, 61
selection (in evol.alg), 47
sequential linear programming, 50, 65, 176
sequential quadratic programming, 51, 67, 177
simplex method
    linear, 7
    nonlinear, 58
simulated annealing, 45
Slater regularity condition, 60
steepest descent, 63
Sturm oscillation theorem, 189
Sturm-Liouville equation, 189
symmetry, 33

tabu search, 47
task partitioning, 160
task scheduling, 160
task synchronization, 160
thermal group, 20
transition matrix, 192
transport equation, *see* neutron transport equation
trust region method, 64

unimodal function, 185

vector computer, 160

warm start, 114



Dankwoord

Toen ik aan mijn huidige chef enkele dagen vrijaf vroeg om dit dankwoord te schrijven —en om nog een aantal zaken rondom het proefschrift te regelen— herinnerde hij mij aan de (blijkbaar internationale) gewoonte om de laatste zin van het dankwoord te wijden aan de personen die het dichtst bij je staan. Inderdaad ben ik mijn familie veel dank verschuldigd. Allereerst mijn ouders, die mij gelegenheid gaven om een wiskundestudie te doen, en me op vele manieren geholpen hebben in mijn Delftse periode. Daarnaast dank ik de rest van mijn familie, die mij verlof gaf voor het spitten van de tuin, terwijl ik wel van de vruchten mag genieten. Ook bedankt voor de keren dat jullie mij van het station kwamen halen als de trein weer eens vertraging had, en voor het leggen van de plavuizen in mijn huis, terwijl ik naar een (achteraf vrij onbelangrijke) conferentie ging. Laat me hier ook Arend-Jan noemen, die zijn vader in de laatste anderhalf jaar hielp door de meeste nachten stil te zijn, ondanks de soms wat magere aandacht overdag. Bijzondere dank gaat naar jouw, Elma, die alleen al door je aanwezigheid rust uitstraalt, die rustig kan luisteren naar technische uiteenzettingen die zelfs voor wiskundige vrienden niet altijd evident zijn, en die altijd zorgt voor de dingen in het leven die niet te maken hebben met wiskunde of computers.

In tegenstelling tot wat misschien verwacht kon worden uit de inleiding, was dit niet mijn laatste zin, want er zijn vele andere mensen die een bijdrage gegeven hebben aan de totstandkoming van dit proefschrift.

In de eerste plaats noem ik mijn begeleiders die me de gelegenheid gaven om dit onderzoek te doen, en me met raad en daad hebben bijgestaan. Het is geen gewoonte dat een promovendus een kamer deelt met zijn begeleider, en het is niet moeilijk om enkele nadelen op te sommen:

1. Als je elkaar zo veel ziet, is het moeilijk om besprekingen te plannen, in het bijzonder als één van de twee het erg druk heeft. En hadden we dan eindelijk een serieuze bespreking dan was er altijd wel de telefoon of iemand aan de deur om ons te storen. Bij tijd en wijle was de meest efficiënte manier van communiceren... via email.

2. Je moet als promovendus uitkijken wat je zegt tegen je computer. Bij een "hoera" is het verleidelijk voor de begeleider om onmiddellijk mee te kijken over je schouder. Dit kan pijnlijk zijn als de "hoera" gebaseerd was op voorbarige conclusies. Ik zal hier niet verder uitweiden over het effect van andere uitroepen...

Ondanks deze opmerkingen, Tamás, wil ik je hartelijk bedanken voor je ondersteuning, voor alle discussies over het project en allerhande andere zaken, voor je lessen hoe om te gaan met onwillige bureaucraten aan de andere kant van de telefoon, en voor je vertrouwen in verschillende situaties.

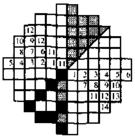
Ook de overige begeleiders wil ik bedanken: Kees Roos, Eduard Hoogenboom en, als waakhond op wat verdere afstand, prof. Freerk Lootsma. Ik ben verder de Reactor Fysica groep van het Interfacultair Reactor Instituut erkentelijk, die dit onderzoek financieel mogelijk heeft gemaakt.

Daarnaast bedank ik René van Geemert voor alle discussies, en voor het wedstrijdelement tussen allerhande zelfgemaakte lokale zoekmethoden enerzijds, en geheeltallige nietlineaire optimalisatie algoritmen anderzijds. Tibor Illés arrived just in time to revive the project with some fresh ideas during the 'second-year dip'. Thanks! Veel dank ben ik verder verschuldigd aan Etienne de Klerk, voor al de bemoedigende en ontmoedigende opmerkingen, voor de nuttige discussies en de prettige samenwerking.

Als volgende noem ik de mede-organisatoren van de workshops on High Performance Optimization. De organisatie kostte meer tijd en inspanning dan ik van te voren had ingeschat, maar het was leuk werk.

Verder bedank ik (in alfabetische volgorde) Bernd, Dima, Erling, Jan, Jiming, Joost, Linda, Lucie, Reza, Vladimir, alle Twaio's, de secretaresses, de systeembeheerders, en alle andere collega's van de vakgroep en daarbuiten. Ook denk ik met plezier aan de andere promovendi van het LNMB. Bijzondere dank gaat naar Henk en Michael voor de 1156 emails die ik terugvond in mijn mailfolders, plus al die mailtjes die ik niet heb bewaard. Ik bedank iedereen die nog niet is genoemd, maar die op welke wijze dan ook mijn periode als promovendus heeft opgefleurd, en zo een bijdrage aan de totstandkoming van dit proefschrift heeft geleverd.

Omdat ik mijn familie als eerste genoemd heb, lijkt dit dankwoord nu af. Als laatste wil ik echter mijn dank uitspreken aan God, mijn Schepper, die me gezegend heeft met de talenten, de gezondheid en dagelijkse krachten om het werk te doen dat geresulteerd heeft in dit proefschrift.



Curriculum Vitae

Arie Quist was born in Tholen at July 22, 1972. After finishing the secondary school at the Calvijn College in Goes in 1990, he studied Applied Mathematics at the Faculty of Technical Mathematics and Informatics of the Delft University of Technology. Starting in 1994, he performed his graduation work at Philips Research N.V. in Eindhoven. In this project he worked on the development and solution of a mathematical optimization model used in the control of MRI medical scanning apparatus.

After obtaining his Master's degree in 1995, he continued working at the optimization group of the Faculty of Technical Mathematics and Informatics of the Delft University of Technology, this time as a PhD-student. The PhD-research was part of a project that was performed in cooperation with, and granted by the Nuclear Reactor Physics Department of the Interfaculty Reactor Institute at the same University. Aim of the project was to investigate whether mathematical optimization techniques can be useful in nuclear reactor core reload pattern optimization. The main results of the research are summarized in this thesis.

Since June 1999 he works for OM Partners N.V. in Brasschaat, Belgium. This company is specialized in long, medium and short-term planning software, and uses mathematical optimization tools for automatic planning. The author is one of the developers of these mathematical optimization tools.