UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Dottorato di Ricerca in
Automatica e Ricerca Operativa

MAT/09

XXI Ciclo

Application-oriented Mixed Integer Non-Linear Programming

Claudia D'Ambrosio

Il Coordinatore	**Il Tutor**
Prof. Claudio Melchiorri	Prof. Andrea Lodi

AA. AA. 2006–2009

Contents

Acknowledgments

I should thank lots of people for the last three years. I apologize in case I forgot to mention someone.

First of all I thank my advisor, Andrea Lodi, who challenged me with this Ph.D. research topic. His contagious enthusiasm, brilliant ideas and helpfulness played a fundamental role in renovating my motivation and interest in research. A special thank goes to Paolo Toth and Silvano Martello. Their suggestions and constant kindness helped to make my Ph.D. a very nice experience. Thanks also to the rest of the group, in particular Daniele Vigo, Alberto Caprara, Michele Monaci, Manuel Iori, Valentina Cacchiani, who always helps me and is also a good friend, Enrico Malaguti, Laura Galli, Andrea Tramontani, Emiliano Traversi.

I thank all the co-authors of the works presented in this thesis, Alberto Borghetti, Cristiana Bragalli, Matteo Fischetti, Antonio Frangioni, Leo Liberti, Jon Lee and Andreas Wächter. I had the chance to work with Jon since 2005, before starting my Ph.D., and I am very grateful to him. I want to thank Jon and Andreas also for the great experience at IBM T.J. Watson Research Center. I learnt a lot from them and working with them is a pleasure. I thank Andreas, together with Pierre Bonami and Alejandro Veen, for the rides and their kindness during my stay in NY.

Un ringraziamento immenso va alla mia famiglia: grazie per avermi appoggiato, supportato, sopportato, condiviso con me tutti i momenti di questo percorso. Ringrazio tutti i miei cari amici, in particolare Claudia e Marco, Giulia, Novi. Infine, mille grazie a Roberto.

Bologna, 12 March 2009 Claudia D'Ambrosio

Keywords

Mixed integer non-linear programming

Non-convex problems

Piecewise linear approximation

Real-world applications

Modeling

List of Figures

List of Tables

Preface

In the most recent years there is a renovate interest for Mixed Integer Non-Linear Programming (MINLP) problems. This can be explained for different reasons: (i) the performance of solvers handling non-linear constraints was largely improved; (ii) the awareness that most of the applications from the real-world can be modeled as an MINLP problem; (iii) the challenging nature of this very general class of problems. It is well-known that MINLP problems are NP-hard because they are the generalization of MILP problems, which are NP-hard themselves. This means that it is very unlikely that a polynomial-time algorithm exists for these problems (unless P = NP). However, MINLPs are, in general, also hard to solve in practice. We address to non-convex MINLPs, i.e. having non-convex continuous relaxations: the presence of non-convexities in the model makes these problems usually even harder to solve.

Until recent years, the standard approach for handling MINLP problems has basically been solving an MILP approximation of it. In particular, linearization of the non-linear constraints can be applied. The optimal solution of the MILP might be neither optimal nor feasible for the original problem, if no assumptions are done on the MINLPs. Another possible approach, if one does not need a proven global optimum, is applying the algorithms tailored for convex MINLPs which can be heuristically used for solving non-convex MINLPs. The third approach to handle non-convexities is, if possible, to reformulate the problem in order to obtain a special case of MINLP problems. The exact reformulation can be applied only for limited cases of non-convex MINLPs and allows to obtain an equivalent linear/convex formulation of the non-convex MINLP. The last approach, involving a larger subset of non-convex MINLPs, is based on the use of convex envelopes or underestimators of the non-convex feasible region. This allows to have a lower bound on the non-convex MINLP optimum that can be used within an algorithm like the widely used Branch-and-Bound specialized versions for Global Optimization. It is clear that, due to the intrinsic complexity from both practical and theoretical viewpoint, these algorithms are usually suitable at solving small to medium size problems.

The aim of this Ph.D. thesis is to give a flavor of different possible approaches that one can study to attack MINLP problems with non-convexities, with a special attention to real-world problems. In Part I of the thesis we introduce the problem and present three special cases of general MINLPs and the most common methods used to solve them. These techniques play a fundamental role in the resolution of general MINLP problems. Then we describe algorithms addressing general MINLPs. Parts II and III contain the main contributions of the Ph.D. thesis. In particular, in Part II four different methods aimed at solving different classes of MINLP problems are presented. More precisely:

In **Chapter 2** we present a Feasibility Pump (FP) algorithm tailored for non-convex Mixed Integer Non-Linear Programming problems. Differences with the previously proposed FP algorithms and difficulties arising from non-convexities in the models are extensively discussed. We show that the algorithm behaves very well with general problems presenting computational results on instances taken from MINLPLib.

In **Chapter 3** we focus on separable non-convex MINLPs, that is where the objective and constraint functions are sums of univariate functions. There are many problems that are already in such a form, or can be brought into such a form via some simple substitutions. We have developed a simple algorithm, implemented at the level of a modeling language (in our case AMPL), to attack such separable problems. First, we identify subintervals of convexity and concavity for the univariate functions using external calls to MATLAB. With such an identification at hand, we develop a convex MINLP relaxation of the problem. We work on each subinterval of convexity and concavity separately, using linear relaxation on only the "concave side" of each function on the subintervals. The subintervals are glued together using binary variables. Next, we repeatedly refine our convex MINLP relaxation by modifying it at the modeling level. Next, by fixing the integer variables in the original non-convex MINLP, and then locally solving the associated non-convex NLP relaxation, we get an upper bound on the global minimum. We present preliminary computational experiments on different instances.

In **Chapter 4** we consider three methods for the piecewise linear approximation of functions of two variables for inclusion within MILP models. The simplest one applies the classical one-variable technique over a discretized set of values of the second independent variable. A more complex approach is based on the definition of triangles in the three-dimensional space. The third method we describe can be seen as an intermediate approach, recently used within an applied context, which appears particularly suitable for MILP modeling. We show that the three approaches do not dominate each other, and give a detailed description of how they can be embedded in a MILP model. Advantages and drawbacks of the three methods are discussed on the basis of some numerical examples.

In **Chapter 5** we present preliminary computational results on heuristics for Mixed Integer Linear Programming. A heuristic for hard MILP problems based on NLP techniques is presented: the peculiarity of our approach to MILP problems is that we reformulate integrality requirements treating them in the non-convex objective function, ending up with a mapping from the MILP feasibility problem to NLP problem(s). For each of these methods, the basic idea and computational results are presented.

Part III of the thesis is devoted to real-world applications: two different problems and approaches to MINLPs are presented, namely Scheduling and Unit Commitment for Hydro-Plants and Water Network Design problems. The results show that each of these different methods has advantages and disadvantages. Thus, typically the method to be adopted to solve a real-world problem should be tailored on the characteristics, structure and size of the problem. In particular:

Chapter 6 deals with a unit commitment problem of a generation company whose aim is to find the optimal scheduling of a multi-unit pump-storage hydro power station, for a short term period in which the electricity prices are forecasted. The problem has a mixed-integer non-linear structure, that makes very hard to handle the corresponding mathematical models. However, modern MILP software tools have reached a high efficiency, both in terms of solution accuracy and computing time. Hence we introduce MILP models of increasing complexity, that allow to accurately represent most of the hydro-electric system characteristics, and turn out to be computationally solvable. In particular, we present a model that takes into account the head effects on power production through an enhanced linearization technique, and turns out to be more general and efficient than those available in the literature. The practical behavior of the models is analyzed through computational experiments on real-world data.

In **Chapter 7** we present a solution method for a water-network optimization problem using a non-convex continuous NLP relaxation and a MINLP search. Our approach employs a relatively simple and accurate model that pays some attention to the requirements of the solvers that we employ. Our view is that in doing so, with the goal of calculating only good feasible solutions, complicated algorithmics can be confined to the MINLP solver. We report successful computational experience using available open-source MINLP software on problems from the literature and on difficult real-world instances.

Part IV of the thesis consists of a brief review on tools commonly used for general MINLP problems. We present the main characteristics of solvers for each special case of MINLP. Then we present solvers for general MINLPs: for each solver a brief description, taken from the manuals, is given together with a schematic table containing the most importart pieces of information, for example, the class of problems they address, the algorithms implemented, the dependencies with external software.

Tools for MINLP, especially open-source software, constituted an integral part of the development of this Ph.D. thesis. Also for this reason Part IV is devoted to this topic. Methods presented in Chapters 4, 5 and 7 were completely developed using open-source solvers (and partially Chapter 3). A notable example of the importance of open-source solvers is given in Chapter 7: we present an algorithm for solving a non-convex MINLP problem, namely the Water Network Design problem, using the open-source software Bonmin. The solver was originally tailored for convex MINLPs. However, some accommodations were made to handle non-convex problems and they were developed and tested in the context of the work presented in Chapter 7, where details on these features can be found.

Part I

Introduction

Chapter 1

Introduction to Mixed Integer Non-Linear Programming Problems and Methods

The (general) Mixed Integer Non-Linear Programming (MINLP) problem which we are interested in has the following form:

MINLP

$$\begin{aligned} \min f(x,y) & & (1.1)\\ g(x,y) &\leq 0 & (1.2)\\ x &\in X \cap \mathbb{Z}^n & (1.3)\\ y &\in Y, & (1.4) \end{aligned}$$

where $f : \mathbb{R}^{n \times p} \to \mathbb{R}$, $g : \mathbb{R}^{n \times p} \to \mathbb{R}^m$, X and Y are two polyhedra of appropriate dimension (including bounds on the variables). We assume that f and g are twice continuously differentiable, but we do not make any other assumption on the characteristics of these functions or their convexity/concavity. In the following, we will call problems of this type non-convex Mixed Integer Non-Linear Programming problems.

Non-convex MINLP problems are NP-hard because they generalize MILP problems which are NP-hard themselves (for a detailed discussion on the complexity of MILPs we refer the reader to Garey and Johnson [61]).

The most complex aspect we need to have in mind when we work with non-convex MINLPs is that its continuous relaxation, i.e. the problem obtained by relaxing the integrality requirement on the x variables, might have (and usually it has) local optima, i.e. solutions which are optimal within a restricted part of the feasible region (neighborhood), but not considering the entire feasible region. This does not happen when f and g are convex (or linear): in these cases the local optima are also global optima, i.e. solutions which are optimal considering the entire feasible region.

To understand more in detail the issue, let us consider the continuous relaxation of the MINLP problem. The first order optimality conditions are necessary, but, only when f and g are convex, they are also sufficient for the stationary point (x, y) which satisfies them to be the global optimum:

$$
\begin{aligned}
g(x,y) &\leq 0 && (1.5)\\
x &\in X && (1.6)\\
y &\in Y && (1.7)\\
\lambda &\geq 0 && (1.8)\\
\nabla f(x,y) + \sum_{i=1}^{m} \lambda_i \nabla g_i(x,y) &= 0 && (1.9)\\
\lambda^T g(x,y) &= 0, && (1.10)
\end{aligned}
$$

where ∇ is the Jacobian of the function and λ are the dual variables, i.e. the variables of the Dual problem of the continuous relaxation of MINLP which is called Primal problem. The Dual problem is formalized as follows:

$$\max_{\lambda \geq 0} [\inf_{x \in X, y \in Y} f(x,y) + \lambda^T g(x,y)], \qquad (1.11)$$

see [17, 97] for details on Duality Theory in NLP. Equations (1.5)-(1.7) are the primal feasibility conditions, equation (1.8) is the dual feasibility condition, equation (1.9) is the stationarity condition and equation (1.10) is the complementarity condition.

These conditions, called also Karush-Kuhn-Tucker (KKT) conditions, assume an important role in algorithms we will present and use in this Ph.D. thesis. For details the reader is referred to Karush [74], Kuhn and Tucker [77].

The aim of this Ph.D. thesis is presenting methods for solving non-convex MINLPs and models for real-world applications of this type. However, in the remaining part of this section, we will present special cases of general MINLP problems because they play an important role in the resolution of the more general problem. Though, this introduction will not cover exhaustively topics concerning Mixed Integer Linear Programming (MILP), Non-Linear Programming (NLP) and Mixed Integer Non-Linear Programming in general, but only give a flavor of the ingredients necessary to fully understand the methods and algorithms which are the contribution of the thesis. For each problem, references will be provided to the interested reader.

1.1 Mixed Integer Linear Programming

A Mixed Integer Linear Programming problem is the special case of the MINLP problem in which functions f and g assume a linear form. It is usually written in the form:

MILP

$$
\begin{aligned}
\min c^T x + d^T y &\\
Ax + By &\leq b\\
x &\in X \cap \mathbb{Z}^n\\
y &\in Y,
\end{aligned}
$$

where A and B are, respectively, the $m \times n$ and the $m \times p$ matrices of coefficients, b is the m-dimensional vector, called right-hand side, c and d are, respectively, the n-dimensional and the p-dimensional vectors of costs. Even if these problems are a special and, in general, easier case with respect to MINLPs, they are NP-hard (see Garey and Johnson [61]). This means that a polynomial algorithm to solve MILP is unlikely to exists, unless P = NP.

Different possible approaches to this problem have been proposed. The most effective ones and extensively used in the modern solvers are Branch-and-Bound (see Land and Doig [78]), cutting planes (see Gomory [64]) and Branch-and-Cut (see Padberg and Rinaldi [102]). These are exact methods: this means that, if an optimal solution for the MILP problem exists, they find it. Otherwise, they prove that such a solution does not exist. In the following we give a brief description of the main ideas of these methods, which, as we will see, are the basis for the algorithms proposed for general MINLPs.

Branch-and-Bound (BB): the first step is solving the continuous relaxation of MILP (i.e. the problem obtained relaxing the integrality constraints on the x variables, LP = $\{\min c^T x + d^T y \mid Ax + By \leq b,\ x \in X,\ y \in Y\}$). Then, given a fractional value x_j^* from the solution of LP (x^*, y^*), the problem is divided into two subproblems, the first where the constraint $x_j \leq \lfloor x_j^* \rfloor$ is added and the second where the constraint $x_j \geq \lfloor x_j^* \rfloor + 1$ is added. Each of these new constraints represents a "branching decision" because the partition of the problem in subproblems is represented with a tree structure, the BB tree. Each subproblem is represented as a node of the BB tree and, from a mathematical viewpoint, has the form:

<u>LP</u>

$$\begin{array}{rcl} \min c^T x + d^T y & & \\ Ax + By & \leq & b \\ x & \in & X \\ y & \in & Y \\ x & \leq & l^k \\ x & \geq & u^k, \end{array}$$

where l^k and u^k are vectors defined so as to mathematically represent the branching decisions taken so far in the previous levels of the BB tree. The process is iterated for each node until the solution of the continuous relaxation of the subproblem is integer feasible or the continuous relaxation is infeasible or the lower bound value of subproblem is not smaller than the current incumbent solution, i.e. the best feasible solution encountered so far. In these three cases, the node is fathomed. The algorithm stops when no node to explore is left, returning the best solution found so far which is proven to be optimal.

Cutting Plane (CP): as in the Branch-and-Bound method, the LP relaxation is solved. Given the fractional LP solution (x^*, y^*), a separation problem is solved, i.e. a problem whose aim is finding a valid linear inequality that cuts off (x^*, y^*), i.e. it is not satisfied by (x^*, y^*). An inequality is valid for the MILP problem if it is satisfied by any integer feasible solution of the problem. Once a valid inequality (cut) is found, it is added to the problem: it makes the LP relaxation tighter and the iterative addition of cuts might lead to an integer solution. Different types of cuts have been studied, for example, Gomory mixed integer cuts, Chvátal-Gomory cuts, mixed integer rounding cuts, rounding cuts, lift-and-project cuts, split cuts, clique cuts (see [40]). Their effectiveness depends on the MILP problem and usually different types of cuts are combined.

Branch-and-Cut (BC): the idea is integrating the two methods described above, merging the advantages of both techniques. Like in BB, at the root node the LP relaxation is solved. If the solution is not integer feasible, a separation problem is solved and, in the case cuts are fonud, they are added to the problem, otherwise a branching decision is performed. This happens also to non-leaf nodes, the LP relaxation correspondent to the node is solved, a separation problem is computed and cuts are added or branch is performed. This method is very effective and, like in CP, different types of cuts can be used.

An important part of the modern solvers, usually integrated with the exact methods, are heuristic methods: their aim is finding rapidly a "good" feasible solution or improving the best solution found so far. No guarantee on the optimality of the solution found is given. Examples of the first class of heuristics are: simple rounding heuristics, Feasibility Pump (see Fischetti et al. [50], Bertacco et al. [16], Achterberg and Berthold [3]). Examples of the second class of heuristics are metaheuristics (see, for example, Glover and Kochenberger [63]), Relaxation Induced Neighborhoods Search (see Danna et al. [43]) and Local Branching (see Fischetti and Lodi [51]).

For a survey of the methods and the development of the software addressed at solving MILPs the reader is referred to the recent paper by Lodi [87], and for a detailed discussion see [2, 18, 19, 66, 95, 104].

1.2 Non-Linear Programming

Another special case of MINLP problems is Non-Linear Programming: the functions f and g are non-linear but $n = 0$, i.e. no variable is required to be integer. The classical NLP problem can be written in the following form:

NLP

$$\begin{aligned} \min f(y) & & & \quad (1.12)\\ g(y) &\leq 0 & & \quad (1.13)\\ y &\in Y. & & \quad (1.14) \end{aligned}$$

Different issues arise when one tries to solve this kind of problems. Some heuristic and exact methods are tailored for a widely studied subclass of these problems: convex NLPs. In this case, the additional assumption is that f and g are convex functions. Some of these methods can be used for more general non-convex NLPs, but no guarantee on the global optimality of the solution is given. In particular, when no assumption on convexity is done, the problem usually has local optimal solutions. In the non-convex case, exact algorithms, i.e. those methods that are guaranteed to find the global solution, are called Global Optimization methods.

In the following we sketch some of the most effective algorithms studied and actually implemented within available NLP solvers (see Section 8):

Line Search: introduced for unconstrained optimization problems with non-linear objective function, it is an iterative method used also as part of methods for constrained NLPs. At each iteration an approximation of the non-linear function is considered and (i) a search direction is decided; (ii) the step length to take along that direction is computed and (iii) the step is taken. Different approaches to decide the direction and the step length are possible (e.g., Steepest descent, Newton, Quasi-Newton, Conjugate Direction methods).

Trust Region: it is a method introduced as an alternative to Line Search. At each iteration, the search of the best point using the approximation is limited into a "trust region", defined with a maximum step length. This approach is motivated by the fact that the approximation of the non-linear function at a given point can be not good far away from that point, then the "trust region" represents the region in which we suppose the approximation is good. Then the direction and the step length which allow the best improvement of the objective function value within the trust region is taken. (The size of the trust region can vary depending on the improvements obtained at the previous iteration.) Also in this case different possible strategies to choose the direction and the step length can be adopted.

Active Set: it is an iterative method for solving NLPs with inequalities. The first step of each iteration is the definition of the active set of constraints, i.e. the inequalities which are strictly satisfied. Considering the surface defined by the constraints within the active set, a move on the surface is decided, identifying the new point for the next iteration. The Simplex algorithm by Dantzig [44] is an Active Set method for solving Linear Programming problems. An effective and widely used special case of Active Set method for NLPs is the Sequential Quadratic Programming (SQP) method. It solves a sequence of Quadratic Programming (QP) problems which approximate the NLP problem at the current point. An example of such an approximation is using the Newton's method to the KKT conditions of the NLP problem. The QP problems are solved using specialized QP solvers.

Interior Point: in contrast to the Active Set methods which at each iteration stay on a surface of the feasible region, the Interior Point methods stay in the strict interior of it. From a mathematical viewpoint, at each iteration, conditions of primal and dual feasibility (1.5)-(1.8) are satisfied and complementarity conditions (1.10) are relaxed. The algorithm aims at reducing the infeasibility of the complementarity constraints.

Penalty and Augmented Lagrangian: it is a method in which the objective function of $\underline{\text{NLP}}$ is redefined in order to take into account both the optimality and the feasibility of a solution adding a term which penalizes infeasible solutions. An example of a Penalty method is the Barrier algorithm which uses an interior type penalty function. When a penalty function is minimized at the solution of the original NLP, it is called exact, i.e. the minimization of the penalty function leads to the optimal solution of the original NLP problem. An example of exact Penalty method is the Augmented Lagrangian method, which makes explicit use of the Lagrange multiplier estimates.

Filter: it is a method in which the two goals (usually competing) which in Penalty methods are casted within the same objective function, i.e. optimality and feasibility, are treated separately. So, a point can become the new iterate point only if it is not dominated by a previous point in terms of optimality and feasibility (concept closely related to Pareto optimality).

For details on the algorithms mentioned above and their convergence, the reader is referred to, for example, [12, 17, 29, 54, 97, 105]. In the context of the contribution of this Ph.D. thesis, the NLP methods and solvers are used as black boxed, i.e. selecting them according to their characteristics and efficiency with respect to the problem, but without changing them.

1.3 Convex Mixed Integer Non-Linear Programming

The third interesting subclass of general MINLP is the convex MINLP. The form of these problems is the same as MINLP, but f and g are convex functions. The immediate and most important consideration derived by this assumption is that each local minimum of the problem is guaranteed to be also a global minimum of the continuous relaxation. This property is exploited in methods studied specifically for this class of problems. In the following we briefly present the most used approaches to solve convex MINLPs, in order to have an idea of the state of the art regarding solution methods of MINLP with convexity properties. Because the general idea of these methods is solving "easier" subproblems of convex MINLPs, we first define three different important subproblems which play a fundamental role in the algorithms we are going to describe.

$\underline{MILP^k}$

$$\begin{aligned}
&\min z\\
&\left.\begin{aligned}
&f(x^k,y^k)+\nabla f(x^k,y^k)^T\begin{bmatrix} x-x^k\\ y-y^k\end{bmatrix}\le z\\
&g(x^k,y^k)+\nabla g(x^k,y^k)^T\begin{bmatrix} x-x^k\\ y-y^k\end{bmatrix}\le 0
\end{aligned}\right\}\quad k=1,\dots,K\\
&x\in X\cap\mathbb{Z}^n\\
&y\in Y,
\end{aligned}$$

where z is an auxiliary variable added in order to have a linear objective function, (x^k, y^k) refers to a specific value of x and y. The original constraint are substituted by their linearization constraints called Outer Approximation cuts.

$\underline{NLP^k}$

$$\begin{aligned}
\min f(x,y)&\\
g(x,y)&\le 0\\
x&\in X\\
y&\in Y\\
x&\le l^k\\
x&\ge u^k,
\end{aligned}$$

where l^k and u^k are, respectively, the lower and upper bound on the integer variables specific of subproblem $\underline{NLP^k}$. This is the continuous subproblem corresponding to a specific node of the Branch-and-Bound tree, i.e. the NLP version of $\underline{LP^k}$. If no branching decision has been taken on a specific variable, say x_j, l_j is equal to $-\infty$ and u_j is equal to $+\infty$ (i.e. the original bounds, included in $x \in X$, are preserved). Otherwise, l_j^k and u_j^k reflect the branching decisions taken so far for variable x_j.

$\underline{NLP_x^k}$

$$\begin{aligned} \min f(x^k, y) & \\ g(x^k, y) &\leq 0 \\ y &\in Y, \end{aligned}$$

where the integer part of the problem is fixed according to the integer vector x^k.

Algorithms studied for convex MINLPs differ basically on how the subproblems involved are defined and used. We present briefly some of the most used algorithms and refer the reader to the exhaustive paper by Grossmann [65].

Branch-and-Bound (BB): the method, originally proposed for MILP problems (see Section 1.1), was adapted for general convex MINLPs by Gupta and Ravindran [70]. The basic difference is that, at each node, an LP subproblem is solved in the first case, an NLP subproblem in the second. We do not discuss specific approaches for branching variable selection, tree exploration strategy, etc. For details the reader is referred to [1, 20, 70, 82].

Outer-Approximation (OA): proposed by Duran and Grossman [45], it exploits the Outer Approximation linearization technique which is "safe" for convex functions, i.e. it does not cut off any solution of the MINLP. It is an iterative method in which, at each iteration k, a $\underline{NLP_x^k}$ and a $\underline{MILP^k}$ subproblem are solved (the vector x^k used in $\underline{NLP_x^k}$ is taken from the solution of $\underline{MILP^k}$). The first subproblem, if feasible, gives an upper bound on the solution of the MINLP and the second subproblem always gives a lower bound. At each iteration the lower and the upper bounds might be improved, in particular the definition of $\underline{MILP^k}$ changes because, at each iteration, OA cuts are added which cut off the solution of the previous iteration. The algorithm ends when the two bounds assume the same value (within a fixed tolerance).

Generalized Benders Decomposition (GBD): like BB, it was first introduced for MILPs (see Benders [15]). Geoffrion [62] adapted the method to convex MINLP problems. It is strongly related to the OA method, the unique difference being the form of the $\underline{MILP^k}$ subproblem. The $\underline{MILP^k}$ of the GBD method is a surrogate relaxation of the one of the OA method and the lower bound given by the OA $\underline{MILP^k}$ is stronger than (i.e. greater or equal to) the one given by the GBD $\underline{MILP^k}$ (for details, see Duran and Grossmann [45]). More precisely, the GBD $\underline{MILP^k}$ constraints are derived from the OA constraints generated only for the active inequalities ($\{i \mid g_i(x^k, y^k) = 0\}$) plus the use of KKT conditions and projection in the x-space:

$$f(x^k, y^k) + \nabla_x f(x^k, y^k)^T (x - x^k) + (\mu^k)[g(x^k, y^k) + \nabla_x g(x^k, y^k)^T (x - x^k)] \leq z$$

where μ^k is the vector of dual variables corresponding to original constraints (1.2) (see [65] for details this relationship). These Lagrangian cuts projected in the x-space are weaker, but the GBD $\underline{MILP^k}$ is easier to solve with respect to the OA $\underline{MILP^k}$. Even if, on average, the number of iterations necessary for the GBD method is bigger than the one for the OA method, the tradeoff among number of iterations and computational effort of each iteration makes sometimes convenient using one or the other approach.

Extended Cutting Plane (ECP): introduced by Westerlund and Pettersson [126], the method is based on the iterative resolution of a $\underline{MILP^k}$ subproblem and, given the optimal solution of $\underline{MILP^k}$ which can be infeasible for $\underline{MINLP}$, the determination of the most violated constraint (or more), whose linearization is added at the next $\underline{MILP^k}$. The given lower bound is decreased at each iteration, but generally a large number of iterations is needed to reach the optimal solution.

LP/NLP based Branch-and-Bound (QG): the method can be seen as the extention of the Branch-and-Cut to convex MINLPs (see Quesada and Grossmann [108]). The idea is solving with BB the $\underline{MILP^k}$ [1] subproblem not multiple times but only once. This is possible if, at each node at which an integer feasible solution is found, the $\underline{NLP^k_x}$ subproblem is solved, OA cuts are then generated and added to the $\underline{MILP^k}$ of the open nodes of the Branch-and-Bound tree.

Hybrid algorithm (Hyb): an enhanced version of QG algorithm was recently developed by Bonami et al. [20]. It is called Hybrid algorithm because it combines BB and OA methods. In particular, the differences with respect to the QG algorithm are that at "some" nodes (not only when an integer solution is found) the $\underline{NLP^k_x}$ subproblem is solved to generate new cuts (like in BB) and local enumerations at some nodes of the tree are performed (it can be seen as performing some iterations of the OA algorithm at some nodes). When the local enumeration is not limited, the Hyb algorithm reconduces to OA, when the $\underline{NLP^k_x}$ is solved at each node, it reconduces to BB.

As for MILP solvers, heuristic algorithms also play an important role within MINLP solvers. Part of the heuristic algorithms studied for MILPs have been adapted for convex MINLP problems. For details about primal heuristics the reader is referred to, for example, [1, 21, 23].

Specific algorithms have been also studied for special cases of convex MINLPs (see, e.g., [56, 69, 109]). Methods which exploit the special structure of the problem are usually much more efficient than general approaches.

1.4 Non-convex Mixed Integer Non-Linear Programming

Coming back to the first model seen in this chapter, $\underline{\text{MINLP}}$, we do not have any convexity assumption on the objective function and the constraints. As discussed, one of the main issues regarding non-convex MINLP problems is that there are, in general, local minima which are not global minima. This issue implies, for example, that, if the NLP solver used to solve $\underline{NLP^k}$ and $\underline{NLP^k_x}$ subproblems does not guarantee that the solution provided is a global optimum (and this is usually the case for the most common NLP solvers, see Chapter 8), feasible and even optimal solutions might be cut off if methods like BB, QG and Hyb of Section 1.4 are used. This happens, for example, when a node is fathomed because of the lower bound (the value of a local minimum can be much worse than the one of the global minimum). This makes these methods, that are exact for convex MINLPs, heuristics for non-convex MINLPs. A second issue involves methods OA, GBD, ECP, QG and Hyb of Section 1.4: the linearization cuts used in these methods are in general not valid for non-convex

[1]The $\underline{MILP^k}$ definition is obtained using the solution (x^0, y^0) of $\underline{NLP^k}$ solved just once for the initialization of the algorithm.

constraints. It means that the linearization cuts might cut off not only infeasible points, but also parts of the feasible region (see Figure 1.1). For this reason, when non-convex constraints are involved, one has to carefully use linearization cuts.

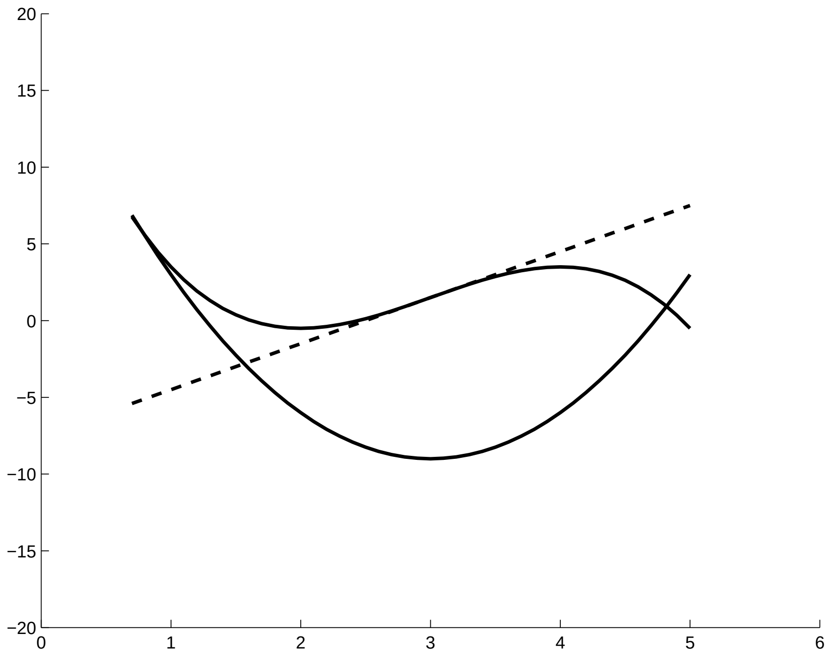


Figure 1.1: Example of "unsafe" linearization cut generated from a non-convex constraint

The first approach to handle non-convexities is, if possible, to reformulate the problem. The exact reformulation can be applied only for limited cases of non-convex MINLPs and allows to obtain an equivalent convex formulation of the non-convex MINLP. All the techniques described in Section 1.3 can then be applied to the reformulated MINLP. For a detailed description of exact reformulations to standard forms, see, for example, Liberti's Ph.D. thesis [83].

The second approach, involving a larger subset of non-convex MINLPs, is based on the use of convex envelopes or underestimators of the non-convex feasible region. This allows to have a lower bound on the non-convex MINLP optimum that can be used within an algorithm like the widely used Branch-and-Bound specialized versions for Global Optimization, e.g., spatial Branch-and-Bound (see [118, 81, 83, 14]), Branch-and-Reduce (see [110, 111, 119]), α-BB (see [6, 5]), Branch-and-Cut (see [76, 100]). The relaxation of the original problem, obtained using convex envelopes or underestimators of the non-convex functions, rMINLP has the form:

$$
\begin{aligned}
&\min z && (1.15)\\
\overline{f}(x,y) &\leq z && (1.16)\\
\overline{g}(x,y) &\leq 0 && (1.17)\\
x &\in X \cap \mathbb{Z}^n && (1.18)\\
y &\in Y, && (1.19)
\end{aligned}
$$

where z is an auxiliary variable added in order to have a linear objective function, $\overline{f} : \mathbb{R}^{n\times p} \longrightarrow$

$\mathbb{R}$ and $\overline{g} : \mathbb{R}^{n \times p} \rightarrow \mathbb{R}^m$ are convex (in some cases, linear) functions and $\overline{f}(x, y) \leq f(x, y)$ and $\overline{g}(x, y) \leq g(x, y)$ within the (x, y) domain.

Explanations on different ways to define functions $\overline{f}(x, y)$ and $\overline{g}(x, y)$ for non-convex functions f and g with specific structure can be found, for example, in [83, 92, 98]. Note anyway that, in general, these techniques apply only for factorable functions, i.e. function which can be expressed as summations and products of univariate functions, which can be reduced and reformulated as predetermined operators for which convex underestimators are known, such as, for example, bilinear, trilinear, fractional terms (see [84, 92, 118]).

The use of underestimators makes the feasible region larger; if the optimal solution of rMINLP is feasible for the non-convex MINLP, then it is also its global optimum. Otherwise, i.e. if the solution of rMINLP is infeasible for MINLP, a refining on the underestimation of the non-convex functions is needed. This is done by branching, not restricted to integer variables but also on continuous ones (see Figure 1.2).

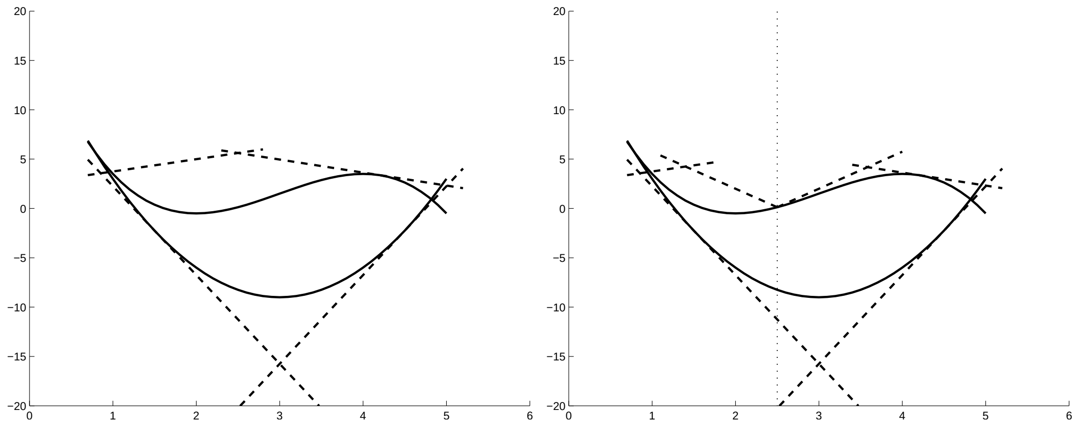


Figure 1.2: Linear underestimators before and after branching on continuous variables

The specialized Branch-and-Bound methods for Global Optimization we mentioned before mainly differ on the branching scheme adopted: (i) branch both on continuous and discrete variables without a prefixed priority; (ii) branch on continuous variables and apply standard techniques for convex MINLPs at each node; (ii) branch on discrete variables until an integer feasible solution is found, then branch on continuous variables.

It is clear that an algorithm of this type is very time-expensive in general. This is the price one has to pay for the guarantee of the global optimality of the solution provided (within a fixed tolerance). Moreover, from an implementation viewpoint, some complex structures are needed. For example, it is necessary to describe the model with symbolic mathematical expressions which is important if the methods rely on tools for the symbolic and/or automatic differentiation. Moreover, in this way it is possible to recognize factors, structures and reformulate the components of the model so as one needs to deal only with standard operators which can be underestimated with well-known techniques. These and other complications arising in the non-convex MINLP software will be discussed more in detail in Chapter 8.

If one does not need a proven global optimum, the algorithms presented in Section 1.3 can be (heuristically) used for solving non-convex MINLPs, i.e. by ignoring the problems explained at the beginning of this section. One example of application of convex methods to non-convex MINLP problems will be presented in Chapter 7. The BB algorithm of the convex

MINLP solver Bonmin [26], modified to limit the effects of non-convexities, was used. Some of these modifications were implemented in Bonmin while studying the application described in Chapter 7 and are now part of the current release.

Also in this case, heuristics studied originally for MILPs have been adapted for non-convex MINLPs. An example is given by a recent work of Liberti et al. [86]. In Chapter 2 we will present a new heuristic algorithm extending to non-convex MINLPs the Feasibility Pump (FP) heuristic ideas for MILPs and convex MINLPs. Using some of the basic ideas of the original FP for solving non-convex MINLPs is not possible for the same reasons we explained before. Also in this case algorithms studied for convex MINLPs encounter problems when applied to non-convex MINLPs. In Chapter 2 we will explain in detail how we can limit these difficulties.

Specific algorithms have been also studied for special cases of non-convex MINLPs (see, e.g., [73, 107, 113]). As for convex MINLPs, methods which exploit the special structure of the problem are usually much more efficient than general approaches. Examples are given in Chapters 3.

1.5 General considerations on MINLPs

Until recent years, the standard approach for handling MINLP problems has basically been solving an MILP approximation of it. In particular, linearization of the non-linear constraints can be applied. Note, however, that this approach differs from, e.g., OA, GBD and ECP presented in Section 1.3 because the linearization is decided before the optimization starts, the definition of the MILP problem is never modified and no NLP (sub)problem resolution is performed. This allows using all the techniques described in Section 1.1, which are in general much more efficient than the methods studied for MINLPs. The optimal solution of the MILP might be neither optimal nor feasible for the original problem, if no assumptions are done of the MINLPs. If, for example, $f(x, y)$ is approximated with a linear objective function, say $\overline{f}(x, y)$, and $g(x, y)$ with linear functions, say $\overline{g}(x, y)$, such that $f(x, y) \geq \overline{f}(x, y)$ and $g(x, y) \geq \overline{g}(x, y)$, the MILP approximation provide a lower bound on the original problem. Note that, also in this case, the optimum of the MILP problem is not guaranteed to be feasible for the original MINLP, but, in case it is feasible for the MINLP problem, we have the guarantee that it is also the global optimum.

In Chapter 4, a method for approximating non-linear functions of two variables is presented: comparisons to the more classical methods like piecewise linear approximation and triangulation are reported. In Chapter 6 we show an example of application in which applying these techniques is successful. We will show when it is convenient applying MINLP techniques in Chapter 7.

Finally, note that the integrality constraint present in Mixed Integer Programming problems can be seen as a source of non-convexity for the problem: it is possible to map the feasibility problem of an MILP problem into an NLP problem. Due to this consideration, we studied NLP-based heuristics for MILP problems: these ideas are presented in Chapter 5.

Part II

Modeling and Solving Non-Convexities

Chapter 2

A Feasibility Pump Heuristic for Non-Convex MINLPs

[1]

2.1 Introduction

Heuristic algorithms have always played a fundamental role in optimization, both as independent tools and as part of general-purpose solvers. Starting from Mixed Integer Linear Programming (MILP), different kinds of heuristics have been proposed: their aim is finding a good feasible solution rapidly or improving the best solution found so far. Within a MILP solver context, both types of heuristics are used. Examples of heuristic algorithms are rounding heuristics, metaheuristics (see, e.g., [63]), Feasibility Pump [50, 16, 3], Local Branching [51] and Relaxation Induced Neighborhoods Search [43]. Even if the heuristic algorithms might find the optimal solution, no guarantee on the optimality is given.

In the most recent years Mixed Integer Non-Linear Programming (MINLP) has become a topic capable of attracting the interest of the research community. This is due from the one side to the continuous improvements of Non-Linear Programming (NLP) solvers and on the other hand to the wide range of real-world applications involving these problems. A special focus has been devoted to convex MINLPs, a class of MINLP problems whose nice properties can be exploited. In particular, under the convexity assumption, any local optimum is also a global optimum of the continuous relaxation and the use of standard linearization cuts like Outer Approximation (OA) cuts [45] is possible, i.e. the generated cuts are valid. Heuristics have been proposed recently also for this class of problems. Basically the ideas originally tailored on MILP problems have been extended to convex MINLPs, for example, Feasibility Pump [21, 1, 23] and diving heuristics [23].

The focus of this chapter is proposing a heuristic algorithm for non-convex MINLPs. These problems are in general very difficult to solve to optimality and, usually, like sometimes also happens for MILP problems, finding any feasible solution is also a very difficult task in practice (besides being NP-hard in theory). For this reason, heuristic algorithms assume a fundamental part of the solving phase. Heuristic algorithms proposed so far for non-convex MINLPs are, for example, Variable Neighborhood Search [86] and Local Branching [93], but

[1]This is a working paper with Antonio Frangioni (DI, University of Pisa), Leo Liberti (LIX, Ecole Polytechnique) and Andrea Lodi (DEIS, University of Bologna).

this field is still highly unexplored. This is mainly due to the difficulties arising from the lack of structures and properties to be exploited for such a general class of problems.

We already mentioned the innovative approach to the feasibility problem for MILPs, called Feasibility Pump, which was introduced by Fischetti et al. [50] for problems with integer variables restricted to be binary and lately extended to general integer by Bertacco et al. [16]. The idea is to iteratively solve subproblems of the original difficult problem with the aim of "pumping" the feasibility in the solution. More precisely, Feasibility Pump solves the continous relaxation of the problem trying to minimize the distance to an integer solution, then rounding the fractional solution obtained. Few years later a similar technique applied to convex MINLPs was proposed by Bonami et al. [21]. In this case, at each iteration, an NLP and an MILP subproblems are solved. The authors also prove the convergence of the algorithm and extend the same result to MINLP problems with non-convex constraints, defining, however, a convex feasible region. More recently Bonami and Goncalves [23] proposed a less time consuming version in which the MILP resolution is substituted by a rounding phase similar to that originally proposed by Fischetti et al. [50] for MILPs.

In this chapter, we propose a Feasibility Pump algorithm for general non-convex MINLPs using ingredients of the previous versions of the algorithm and adapting them in order to remove assumptions about any special structure of the problem. The remainder of the chapter is organized as follows. In Section 2.2 we present the structure of the algorithm, then we describe in detail each part of it. Details on algorithm (implementation) issues are given in Section 2.3. In Section 2.4 we present computational results on MINLPLib instances. Finally, in Section 2.5, we draw conclusions and discuss future work directions.

2.2 The algorithm

The problem which we address is the non-convex MINLP problem of the form:

$$
\begin{aligned}
(P) \qquad\qquad & \min f(x,y) && (2.1)\\
& g(x,y) \leq 0 && (2.2)\\
& x \in X \cap \mathbb{Z}^n && (2.3)\\
& y \in Y, && (2.4)
\end{aligned}
$$

where X and Y are two polyhedra of appropriate dimension (including bounds on the variables), $f : \mathbb{R}^{n+p} \rightarrow \mathbb{R}$ is *convex*, but $g : \mathbb{R}^{n+p} \rightarrow \mathbb{R}^m$ is *non-convex*. We will denote by $\mathcal{P} = \{ \ (x,y) \mid g(x,y) \leq 0 \ \} \subseteq \mathbb{R}^{n+p}$ the (non-convex) feasible region of the continuous relaxation of the problem, by $\mathcal{X}$ the set $\{1,\ldots,n\}$ and by $\mathcal{Y}$ the set $\{1,\ldots,p\}$. We will also denote by $\mathcal{NC} \subseteq \{1,\ldots,m\}$ the subset of (indices of) non-convex constraints, so that $\mathcal{C} = \{1,\ldots,m\} \setminus \mathcal{NC}$ is the set of (indices of) "ordinary" convex constraints. Note that the convexity assumption on the objective function f can be taken without loss of generality; one can always introduce a further variable v, to be put alone in the objective function, and add the $(m+1)$th constraint $f(x,y) - v \leq 0$ to deal with the case where f is non-convex.

The problem (P) presents two sources of non-convexities:

1. integrality requirements on x variables;

2. constraints $g_j(x,y) \leq 0$ with $j \in \mathcal{NC}$, defining a non-convex feasible region, even if we do not consider the integrality requirements on x variables.

The basic idea of Feasibility Pump is decomposing the original problem in two easier subproblems, one obtained relaxing integrality constraints, the other relaxing "complicated" constraints. At each iteration a pair of solutions $(\bar{x}, \bar{y})$ and $(\hat{x}, \hat{y})$ is computed, the solution of the first subproblem and the second one, respectively. The aim of the algorithm is making the trajectories of the two solutions converge to a unique point, satisfying all the constraints and the integrality requirements (see Algorithm 1).

Algorithm 1 The general scheme of Feasibility Pump

```
1: i=0;
2: while $(((\hat{x}^i, \hat{y}^i) \neq (\bar{x}^i, \bar{y}^i)) \wedge$ time limit) do
3:     Solve the problem $(P1)$ obtained relaxing integrality requirements (using all other constraints) and minimizing a "distance" with respect to $(\hat{x}^i, \hat{y}^i)$;
4:     Solve the problem $(P2)$ obtained relaxing "complicated" constraints (using the integrality requirements) minimizing a "distance" with respect to $(\bar{x}^i, \bar{y}^i)$;
5:     i++;
6: end while
```

When the original problem (P) is a MILP, $(P1)$ is simply the LP relaxation of the problem and solving $(P2)$ corresponds to a rounding of the fractional solution of $(P1)$ (all the constraints are relaxed, see Fischetti et al. [50]). When the original problem (P) is a MINLP, $(P1)$ is the NLP relaxation of the problem and $(P2)$ a MILP relaxation of (P). If MINLP is convex, i.e. $\mathcal{NC} = \emptyset$, we know that $(P1)$ is convex too and it can ideally be solved to global optimality and that $(P2)$ can be "safely" defined as the Outer Approximation of (P) (see, e.g., Bonami et al. [21]) or a rounding phase (see Bonami and Goncalves [23]).

When $\mathcal{NC} \neq \emptyset$, things get more complicated:

the solution provided by the NLP solver for problem $(P1)$ might be only a local minimum instead of a global one. Suppose that the global solution of problem $(P1)$ value is 0 (i.e. it is an integer feasible solution), but the solver computes a local solution of value greater than 0. The OA cut generated from the local solution might mistakenly cut the integer feasible solution.

Outer Approximation cuts can cut off feasible solutions of (P), so these cuts can be added to problem $(P2)$ only if generated from constraints with "special characteristics" (which will be presented in detail in Section 2.2.2). This difficulty has implications also on the possibility of cycling of the algorithm.

We will discuss these two issues and how we limit their impact in the next two sections.

2.2.1 Subproblem $(P1)$

At iteration i subproblem $(P1)$, denoted as $(P1)^i$, has the form:

$$\min ||x - \hat{x}^i|| \tag{2.5}$$

$$g(x, y) \leq 0 \tag{2.6}$$

where $(\hat{x}^i,\ \hat{y}^i)$ is the solution of subproblem $(P2)^i$ (see Section 2.2.2). The motivation for solving problem $(P1)^i$ is twofold: (i) testing the compatibility of values $\hat{x}^i$ with a feasible solution of problem (P) (such a solution exists if the solution value of $(P1)^i$ is 0); (ii) if no feasible solution with x variables assuming values $\hat{x}^i$, a feasible solution for $\mathcal{P}$ is computed minimizing the distance $||x - \hat{x}^i||$. As anticipated in the previous section, when $g(x, y)$ are non-convex functions $(P1)^i$, has, in general, local optima, i.e. solutions which are optimal considering a restricted part of feasible region (neighborhood). Available NLP solvers usually do not guarantee to provide the global optimum, i.e. an optimal solution with respect to the whole feasible region. Moreover, solving a non-convex NLP to global optimality is in general very time consuming. Then, the first choice was to give up trying to solve $(P1)^i$ to global optimality. The consequences of this choice are that, when a local optimum is provided as solution of $(P1)^i$ and its value is greater than 0, there might be a solution of (P) with values $\hat{x}^i$, i.e. the globally optimal solution might have value 0. In this case we would, mistakenly, cut off a feasible solution of (P). To limit this possibility we decided to divide step 3 of Algorithm 1 in two parts:

1. Solve $(P1)^i$ to local optimality, but multiple times, i.e. using randomly generated starting points;

2. If no solution was found, then solve $(P1fix)^i$:

$$\min f(\hat{x}^i, y) \tag{2.7}$$

$$g(\hat{x}^i, y) \leq 0 \tag{2.8}$$

   Note that objective function (2.5) is useless when variables x are fixed to $\hat{x}^i$, so we can use the original objective function or, alternatively, a null function or a function which helps the NLP solver to reach feasibility.

The solution proposed does not give any guarantee that the global optimum will be found and, consequentely, that no feasible solution of (P) will be ignored, but, since we propose a heuristic algorithm, we consider this simplification as a good compromise. Note, however, that for some classes of non-convex MINLP the solution does the job. Consider, for example, a problem (P) that, once variables x are fixed, is convex: in this case solving problem $(P1fix)^i$ would provide the global optimum. In Section 2.4 we will provide details on the computational behavior of the proposed solution.

2.2.2 Subproblem $(P2)$

At iteration i subproblem $(P2)$, denoted as $(P2)^i$, has the form:

$$\min ||x - \bar{x}^{i-1}|| \tag{2.9}$$

$$g_j(\bar{x}^k, \bar{y}^k) + \nabla g_j(\bar{x}^k, \bar{y}^k)^T \begin{bmatrix} x - \bar{x}^k \\ y - \bar{y}^k \end{bmatrix} \leq 0 \quad k = 1, \dots, i-1; j \in M^k \tag{2.10}$$

$$x \in \mathbb{Z}^n \tag{2.11}$$

$$y \in \mathbb{R}^p, \tag{2.12}$$

where $(\bar{x}^{i-1}, \bar{y}^{i-1})$ is the solution of subproblem $(P1)^{i-1}$ and $M^k \subseteq \{1, \dots, m\}$ is the set of (indices of) constraints from which OA cuts are generated from point $(\bar{x}^k, \bar{y}^k)$. We limit the OA cuts added to $(P2)$ because, when non-convex constraints are involved, not all the possible OA cuts generated are "safe", i.e. do not cut off feasible solutions of (P) (see Figure 6.1).

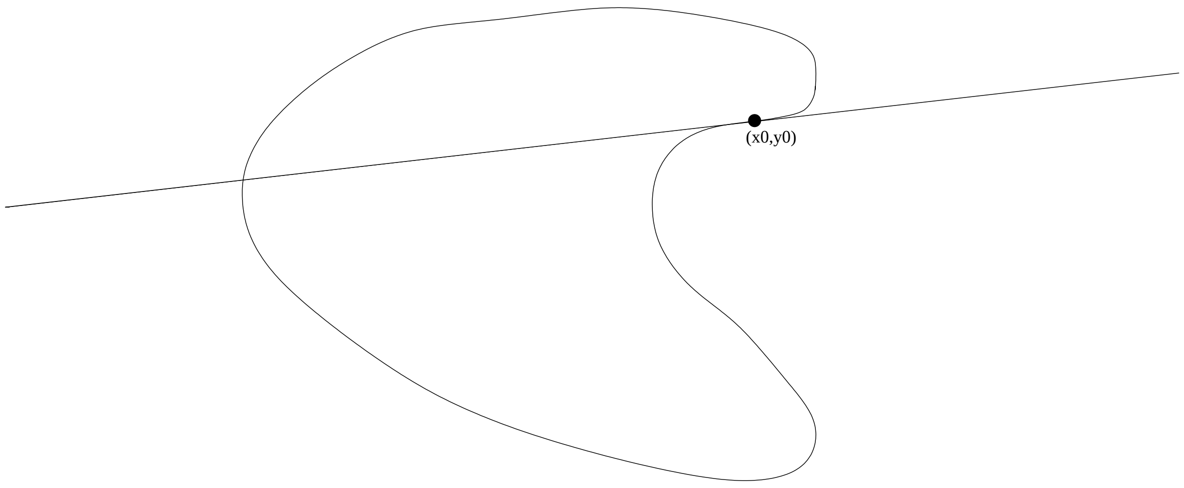


Figure 2.1: Outer Approximation constraint cutting off part of the non-convex feasible region.

When the OA cut is generated from a convex and tight constraint $g_m(x, y)$ it is valid. Indeed, let z^* be the feasible solution of step 2.A and let $g_j(z) \leq 0$ be the convex constraint that is tight to z^*. The OA constraint would be: $\nabla g_j(z^*)^T(z - z^*) \leq 0$. Note that since g_j is convex, this property holds $g_j(x) + \nabla g_j(x)^T(y - x) \leq g_j(y)$ for each x,y in the domain where g_j is convex. Then, $\forall z \in \mathcal{P}$, $g_j(z^*) + \nabla g_j(z^*)^T(z - z^*) \leq g_j(z)$. Since $g_j(z) \leq 0$ is tight to z^*, we have $g_j(z^*) = 0$ and $\nabla g_j(z^*)^T(z - z^*) \leq g_j(z)$. $\forall z \in \mathcal{P}$, $g_j(z) \leq 0$, then $\nabla g_j(z^*)^T(z - z^*) \leq 0$ is a valid cut for the original problem.

The problem with this involves basically two issues, one from a practical, the other from the theoretical viewpoint. The first issue is that discriminating convex and non-convex constraints is a hard task in practice. We will describe in Section 2.4 how we simplified this on the implementation side. The second issue is that Outer Approximation cuts play a fundamental role on convergence of the algorithm, i.e. if at one iteration no OA cut can be added, the algorithm may cycle. However, even if an OA cut is added, there is no guarantee that it would cut off the solution of the previous iteration, see, for example, Figure 2.2. In the figure, the non-linear feasible region and its current linear approximation. The solution of subproblem $(P1)$ is $\bar{x}$ and, in this case, only one Outer Approximation can be generated, the one corresponding to the tight and convex constraint. However, this OA cut does not cut off solution $\hat{x}$, but, in the example, the FP would not cycle, as the MILP at the next iteration would not pick out $\hat{x}$. This shows that there is a distinction between cutting off and cycling. However

We propose two solutions to this theoretical issue which will be described in the next two sections.

"No-good" cuts

One idea could be adding a constraint of the form:

$$\|x - \hat{x}\| \geq \varepsilon \ , \tag{2.13}$$

being valid for all feasible solutions of $(P1)$, if valid for all integer feasible solutions, too. So it can be added to $(P2)$ and it cuts off $\hat{x}$ (of the previous iteration). The problem with constraint

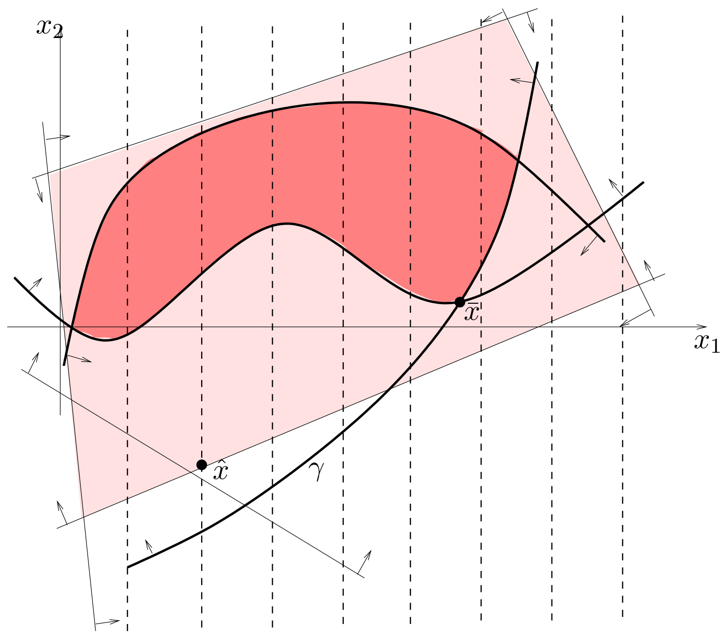


Figure 2.2: The convex constraint γ does not cut off $\hat{x}$, so nor does any OA linearization at $\bar{x}$.

(2.13) is that it is non-convex. However, there are different ways to transform constraint (2.13) in a linear constraint. In general they are quite inefficient, but for some special cases, like the (important) case in which $x \in \{0,1\}^n$, constraint (2.13) can be transformed in:

$$\sum_{j:\hat{x}_j^i=0} x_j + \sum_{j:\hat{x}_j^i=1} (1-x_j) \geq 1 \tag{2.14}$$

without requiring any additional variable or constraint. Defining the norm of constraint (2.13) as $\|\|_1$ and because $\hat{x}_j$ can be only 0 or 1, in the first case $\|x_j - \hat{x}_j\| = x_j$, in the latter $\|x_j - \hat{x}_j\| = 1 - x_j$, and we have, for $\varepsilon = 1$, equation (2.14). Exploiting this idea one can generalize the "no-good" cut valid for the binary case to the general integer case. The "no-good" cut for general integer variables reads as follows:

$$\sum_{j\in\mathcal{X}:\hat{x}_j=l_j} (x_j - l_j) + \sum_{j\in\mathcal{X}:\hat{x}_j=u_j} (u_j - x_j) + \sum_{j\in\mathcal{X}:l_j<\hat{x}_j<u_j} (x_j^+ + x_j^-) \geq 1, \tag{2.15}$$

where, for all $j \in \mathcal{X}$, we need the following additional constraints and variables:

$$x_j = \hat{x} + x_j^+ - x_j^- \tag{2.16}$$

$$x_j^+ \geq z_j(u_j - l_j) \tag{2.17}$$

$$x_j^- \geq (1-z_j)(u_j - l_j) \tag{2.18}$$

$$z_j \in \{0,1\}. \tag{2.19}$$

This leads to an inefficient way to handle the "no-good" cut, because $2n$ additional continuous variables, n additional binary variables and $3n+1$ additional equations are needed.

This MILP formulation of the "no-good" cut for general integer can be seen as the interval-gradient cut of constraint (2.13) using $\|\|_1$ and $\varepsilon = 1$.

In the following we present some considerations about the relationship between the interval-gradient cut (see [98]) and the "no-good" cut proposed (2.15)-(2.19). Suppose we have a non-convex constraint $g(x) \leq 0$ with $x \in [\underline{x}, \overline{x}]$ and that $[\underline{d}, \overline{d}]$ is the interval-gradient of g over $[\underline{x}, \overline{x}]$, i.e. $\nabla g(x) \in [\underline{d}, \overline{d}]$ for $x \in [\underline{x}, \overline{x}]$. The interval-gradient cut generated from this constraint with respect to a point $\hat{x}$ is:

$$\underline{g}(x) = g(\hat{x}) + \min_{d \in [\underline{d}, \overline{d}]} d^T(x - \hat{x}) \leq 0. \tag{2.20}$$

(Here we are exploiting the following property: $\underline{g}(x) \leq g(x) \leq 0$, then we know the cut is valid.) Equation (2.20) can be reformulated with the following MILP model:

$$g(\hat{x}) + \sum_{j \in \mathcal{X}} (\underline{d}x^+ - \overline{d}x^-) \leq 0 \tag{2.21}$$

$$x - \hat{x} = x^+ - x^- \tag{2.22}$$

$$x_j^+ \leq z_j(\overline{x}_j - \underline{x}_j) \qquad j \in \mathcal{X} \tag{2.23}$$

$$x_j^- \leq (1 - z_j)(\overline{x}_j - \underline{x}_j) \qquad j \in \mathcal{X} \tag{2.24}$$

$$x^+ \geq 0, x^- \geq 0 \tag{2.25}$$

$$z \in \{0,1\}^n \tag{2.26}$$

with the cost of $2n$ additional continuous variables, n additional binary variables and $3n + 1$ additional constraints. Now, consider equation (2.13). It is non-convex and we try to generate an interval-gradient cut with respect to point $\hat{x}$. We first transform equation (2.13) in this way (using $\|\|_1$ and $\varepsilon = 1$):

$$\underline{g}(x) \leq g(x) = -\sum_{j \in \mathcal{X}} |x_j - \hat{x}_j| \leq -1 \ . \tag{2.27}$$

First consideration: $g(\hat{x}) = 0$. Now let analyze a particular index $j \in \mathcal{X}$. Three cases are possible:

1. $\hat{x}_j = \underline{x}_j$: this implies that $-|x_j - \hat{x}_j| = \hat{x}_j - x_j$ and $\underline{d} = \overline{d} = -1$. The term $(\underline{d}x_j^+ - \overline{d}x_j^-)$ become $-x_j^+ + x_j^- = -x_j + \hat{x}_j = \underline{x}_j - x_j$.

2. $\hat{x}_j = \overline{x}_j$: this implies that $-|x_j - \hat{x}_j| = x_j - \hat{x}_j$ and $\underline{d} = \overline{d} = 1$. The term $(\underline{d}x_j^+ - \overline{d}x_j^-)$ become $x_j^+ - x_j^- = x_j - +\hat{x}_j = x_j - \overline{x}_j$.

3. $\underline{x}_j \leq \hat{x}_j \leq \overline{x}_j$: this implies that $\underline{d} = -1$ and $\overline{d} = 1$. The term $(\underline{d}x_j^+ - \overline{d}x_j^-)$ become $-(x_j^+ + x_j^-)$.

We can then simplify equation (2.21) in this way:

$$\sum_{j \in \mathcal{X} : \hat{x}_j = \underline{x}_j} (\underline{x}_j - x_j) + \sum_{j \in \mathcal{X} : \hat{x}_j = \overline{x}_j} (x_j - \overline{x}) + \sum_{j \in \mathcal{X} : \underline{x}_j < \hat{x}_j < \overline{x}_j} (-x_j^+ - x_j^-) \leq -1 \tag{2.28}$$

changing the sign and completing the MILP model:

$$\sum_{j\in\mathcal{X}:\hat{x}_j=\underline{x}_j}(x_j-\underline{x}_j)+\sum_{j\in\mathcal{X}:\hat{x}_j=\overline{x}_j}(\overline{x}-x_j)+\sum_{j\in\mathcal{X}:\underline{x}_j<\hat{x}_j<\overline{x}_j}(x_j^+ + x_j^-)\geq 1 \tag{2.29}$$

$$x_j^+\leq z_j(\overline{x}_j-\underline{x}_j)\qquad j\in\mathcal{X} \tag{2.30}$$

$$x_j^-\leq (1-z_j)(\overline{x}_j-\underline{x}_j)\qquad j\in\mathcal{X} \tag{2.31}$$

$$x^+\geq 0, x^-\geq 0 \tag{2.32}$$

$$z\in\{0,1\}^n \tag{2.33}$$

which is exactly the "no-good" cut for general integer.
In the following we present the details of how to linearize the "no-good" cut (2.13) in the general integer case. In particular we considered two cases:

1. Using $\|\cdot\|_\infty$ for problem $(P1)$;
2. Using $\|\cdot\|_1$ for problem $(P1)$.

We explicitly define the NLP problems for the two cases:

$$(NLP)\qquad (\bar{x},\bar{y})=\text{argmin}\{\ \|x-\hat{x}\|_\infty\ :\ g(x,y)\leq 0\ \}=$$
$$\text{argmin}\{\ \varepsilon\ :\ -\varepsilon\leq x_i-\hat{x}_i\leq\varepsilon\quad\forall i,\ \varepsilon\geq 0,\ g(x,y)\leq 0\ \}$$

$$(NLP)\qquad (\bar{x},\bar{y})=\text{argmin}\{\ \|x-\hat{x}\|_1\ :\ g(x,y)\leq 0\ \}=$$
$$\text{argmin}\{\ \varepsilon=\sum_i v_i\ :\ -v_i\leq x_i-\hat{x}_i\leq v_i\quad\forall i,\ v\geq 0,\ g(x,y)\leq 0\ \}.$$

In both the cases, if the objective function value of the optimal solution of NLP is equal to 0, we have an integer and feasible solution that is $\hat{x}$. If this is not the case, we have to solve the MILP problem, but we want to add something that avoid the possible cycling. If there are some convex constraints that are tight for $(\bar{x},\bar{y})$, we can generate the OA constraints that is added to the MILP problem of the previous iteration and we will get a new $(\hat{x}^k,\hat{y}^k)$. If there is no convex constraint that is tight, we will solve one of these two problems (resp, for the $\|\cdot\|_\infty$ and $\|\cdot\|_1$ cases):

$$(MILP^k)\quad (\hat{x}^k,\hat{y}^k)=\text{argmin}\{\ \|x-\bar{x}\|\ :\ g_j(\bar{x}^k,\bar{y}^k)+\nabla g_j(\bar{x}^k,\bar{y}^k)^T\begin{bmatrix} x-\bar{x}^k\\ y-\bar{y}^k\end{bmatrix}\leq 0$$
$$k=1,\dots,i-1; j\in M^k,\ \lceil\varepsilon\rceil\leq x_i-\hat{x}_i^{k-1}+M(1-z_i)\quad\forall i,$$
$$\lceil\varepsilon\rceil\leq\hat{x}_i^{k-1}-x_i+M(1-z_i')\quad\forall i,\ \sum_i z_i+\sum_i z_i'\geq 1,\ z\in\{0,1\}^p,\ z'\in\{0,1\}^p\ \}$$

(Intuitively, if z_i (or z_i') is equal to 1, the correspondent constraint is active and we impose that the component i is changed wrt $\hat{x}^{k-1}$)

$$(MILP^k)\quad (\hat{x}^k,\hat{y}^k)=\text{argmin}\{\ \|x-\bar{x}\|\ :\ g_j(\bar{x}^k,\bar{y}^k)+\nabla g_j(\bar{x}^k,\bar{y}^k)^T\begin{bmatrix} x-\bar{x}^k\\ y-\bar{y}^k\end{bmatrix}\leq 0$$
$$k=1,\dots,i-1; j\in M^k,\ v_i\leq x_i-\hat{x}_i^{k-1}+Mz_i\quad\forall i,$$
$$v_i\leq\hat{x}_i^{k-1}-x_i+M(1-z_i)\quad\forall i,\ \sum_i v_i\geq\lceil\varepsilon\rceil,\ z\in\{0,1\}^p\ \}$$

(Intuitively, if v_i is greater than 0, one of the 2 correspondent constraints is active and we impose that the component i is changed wrt $\hat{x}^{k-1}$. The minimum total change is $\lceil \varepsilon \rceil$).
where ε is the objective function value of the optimal solution of NLP, $\hat{x}^{k-1}$ is the optimal solution of MILP at the previous iteration, M is the big M coefficient that has to be defined in a clever way.

Tabu list

An alternative way, which do not involve modifications of the model such as introducing additional variables and complicated constraints, is using a tabu list of the last solutions computed by $(P2)$. From a practical viewpoint this is possible using a feature available within the MILP solver Ilog Cplex [71] called "callback". The one we are interested in is the "incumbent callback", a tool which allows the user to define a function which is called during the execution of the Branch-and-Bound whenever Cplex finds a new integer feasible solution. Within the callback function the integer feasible solution computed by the solver is available. The integer part of the solution is compared with the one of the solutions in the tabu list and, only if the solution has a tollerable diversity with respect to the forbidden solutions, it is accepted. Otherwise it is discarted and the Branch-and-Bound execution continues. In this way, even if the same solution can be obtained in two consequent iteration, the algorithm discarts it, then it does not cycle. It is a simple idea which works both with binary and with general integer variables. The diversity of two solutions, say, $\hat{x}^1$ and $\hat{x}^2$, is computed in the following way:

$$\sum_{j \in \mathcal{X}} |\hat{x}^1_j - \hat{x}^2_j|,$$

and two solutions are different if the above sum is not 0. Note that the continuous part of the solution does not influence the diversity measure.

2.2.3 The resulting algorithm

The general scheme of the algorithm proposed is described by Algorithm 2. The resolution of problem $(P1)$ is represented by steps 10-27 and the resolution of problem $(P2)$ is represented by steps 28-40. At step 23, a restriction of problem (P) is solved. This restriction is "generally" a non-convex NLP we solve to local optimality, if we have no assumption on the structure of the problem. The objective function problem of step 34 is $\|x - \bar{x}^i\|_1$. Finally note that, if use_tabu_list is 0, TL is $\emptyset$ and no integer solution will be rejected (the "no-good" cuts do the job).

In the next section we present details on the implementation of the algorithm.

2.3 Software structure

The algorithm was implemented within the AMPL environment [55]. We choose to use this framework to be flexible with respect to the solver we want to use in the different phases of the proposed algorithm. In practice, the user can select the preferred solver to solve NLPs or MILPs, exploiting advantages of the chosen solver.

The input is composed of two files: (i) the mod file where the model of the instance is implemented, called "fpminlp.mod"; (ii) the file "parameter.txt", in which one can de-

Algorithm 2 The general scheme of the proposed algorithm

```
Require: time_limit; use_tabu_list; use_no_good_cuts; use_fix_int_vars; perc_time_NLP;
 1: Getting information about the model;
 2: Init solution $(\hat{x}^0, \hat{y}^0)$ and the parameters;
 3: i = 0; start_time = time();
 4: if use_tabu_list == 1 then
 5:    TL = $\{(\hat{x}^0, \hat{y}^0)\}$;
 6: else
 7:    TL = $\emptyset$;
 8: end if
 9: while start_time+time_limit>time() do
10:    count_NLP = 0; start_time_NLP = time();
11:    while start_time_NLP+(time_limit perc_time_NLP)>time() do
12:       Select randomly the starting point for the NLP solver (within the variables bound
          ranges);
13:       Solve the $(P1)^i$ $(\bar{x}^i, \bar{y}^i) = \text{argmin}\{ \|x - \hat{x}^i\|_2 \ : \ g(x,y) \leq 0 \}$;
14:       count_NLP++;
15:       if $\|\bar{x}^i - \hat{x}^i\|_2 == 0$ then
16:          return $(\bar{x}^i, \bar{y}^i)$;
17:       end if
18:       if (a feasible solution for $(P1)^i$ was found) then
19:          break;
20:       end if
21:    end while
22:    if (a feasible solution for $(P1)^i$ was not found)&&(use_fix_int_vars == 1) then
23:       Solve $(P1_{fix})^i$ $(\bar{x}^i, \bar{y}^i) = \text{argmin}\{ f(\hat{x}^i, y) \ : \ g(\hat{x}^i, y) \leq 0\}$;
24:       if $\|\bar{x}^i - \hat{x}^i\|_2 == 0$ then
25:          return $(\hat{x}^i, \hat{y}^i)$;
26:       end if
27:    end if
28:    if (at least one OA constraint can be generated) && (the corresponding OA constraint
       cuts off $(\hat{x}^i, \hat{y}^i)$ ) then
29:       Amend problem $(P2)$ with an OA linear constraint;
30:    end if
31:    if use_no_good_cuts == 1 then
32:       Add an appropriate “no-good” cut to $(P2)$;
33:    end if
34:    Solve $(P2)^i$ without accepting solutions $\in$ TL and get the new solution $(\hat{x}^{i+1}, \hat{y}^{i+1})$;
35:    if $(\hat{x}^{i+1}, \hat{y}^{i+1})$ is feasible for (P) then
36:       return $(\hat{x}^{i+1}, \hat{y}^{i+1})$;
37:    end if
38:    if use_tabu_list == 1 then
39:       Update TL;
40:    end if
41:    i++;
42: end while;
43: if time() > time_limit then
44:    return false;
45: end if
```

fine parameters of step 2, the NLP solver (*NLP_solver*), the precision (*_FP_epsilon* and *_FP_perc_infeas_allowed*), the level of verbosity (*VERBOSE*).

To solve problem ($P1$) and the restriction of step 23, we use the NLP solver as a black-box. We use solvers directly providing an AMPL interface, which is anyway usually the case for the most common and efficient NLP solvers.

To solve problem ($P2$), if use_tabu_list is 0, we use an MILP solver as a black-box. The interaction with the MILP solver is done in the same way of NLP solver for ($P1$). When use_tabu_list is 1, we use an executable called "tabucplex" to solve problem ($P2$). As anticipated in Section 2.2.2, we implemented this modification to a standard Branch-and-Bound method within the Ilog Cplex environment, exploiting the so-called callbacks. This file reads and stores data of an input file produced within the AMPL framework which contains the tabu list, i.e. the list of the "forbidden" solutions. Then the Branch-and-Bound starts: its execution is standard until an integer feasible solution is found. Every time an integer feasible solution is found, the specialized incumbent callback is called. The aim of this function is to check if the solution found is within those solutions in the tabu list, i.e. it was provided as problem ($P2$) solution in one of the previous iterations. If this is the case, the solution is rejected, otherwise the solution is accepted. In any case, the execution of the Branch-and-Bound continues until the optimal solution, excluding the forbidden ones, is found or a time limit is reached.

Another tool we extensively used is a new solver/reformulator called Rose (Reformulation/Optimization Software Engine, see [85]), of which we exploited the following nice features:

1. Analyzing the model, i.e. getting information about non-linearity and convexity of the constraints and integrality requirements of the variables: necessary at step 1. These parameters are provided by Rose as AMPL suffixes.

2. Analyzing feasibility of the solution: necessary after steps 13, 23 and for step 35 to verify feasibility of the provided solutions. Also in this case parameters are provided by Rose as AMPL suffixes.

3. Generating the Outer Approximation cuts: necessary at step 29. Cuts are written in a file which is then included within the AMPL framework.

Actually some of these features were implemented within the context of this work. Note that information about the convexity of the constraints are hard to compute: in particular, Rose gives information about the "evidently convex"/"evidently concave" constraints using the expression tree, properties of convex/concave functions and basic expressions (see [85] for details). In practice, a non-convex constraint is always identified, a convex constraint can be treated as non-convex constraint, but the information provided is in any case "safe" for our purposes, i.e. we generate OA cuts only from constraints which are "certified" to be convex.

An obvious modification of the algorithm proposed is considering the original objective function to improve the provided solution quality as done, for example, in [50] and [21].

2.4 Computational results

In this section preliminary computational results are presented on an Intel Xeon 2.4 GHz with 8 GB RAM running Linux. We stop the algorithm after the first MINLP feasible solution was found (or the time limit is reached). The parameters were set in the following way:

time_limit = 2 hours;

use_tabu_list = 1;

use_no_good_cuts = 0;

use_fix_int_vars = 1;

perc_time_NLP = 0.05;

_FP_epsilon = 1e-6;

_FP_perc_infeas_allowed = 0.001;

The NLP solver used is Ipopt 3.5 trunk [123] and the problems solved are 243 instances taken from MINLPLib [32] (the ones used in [86] minus `oil` and `oil2` because function log10 is not supported by Rose). Tabucplex uses the Callable library of Ilog Cplex 11.0.

We found an MINLP feasible solution for 200 instances (see Table 2.1). The average CPU time is 174.45 seconds. For 28 of these instances the solution found is also the best known

Table 2.1: Instances for which a feasible solution was found within the time limit

alan	ex1223a	fo7_ar2_1	m7_ar2_1	nuclearvb	nvs22	sep1	st_test6
batchdes	ex1223b	fo7_ar25_1	m7_ar25_1	nuclearvc	nvs23	space25a	st_test8
batch	ex1223	fo7_ar3_1	m7_ar3_1	nuclearvd	nvs24	space25	st_testgr1
contvar	ex1224	fo7_ar4_1	m7_ar4_1	nuclearve	o7_2	spectra2	st_testgr3
csched1a	ex1225	fo7_ar5_1	m7_ar5_1	nuclearvf	o7_ar2_1	spring	st_testph4
csched1	ex1226	fo7	m7	nvs01	o7_ar25_1	st_e13	synheat
csched2a	ex1233	fo8_ar2_1	mbtd	nvs02	o7_ar3_1	st_e14	synthes1
csched2	ex1243	fo8_ar4_1	meanvarx	nvs03	o7_ar4_1	st_e15	synthes2
deb6	ex1244	fo8_ar5_1	minlphix	nvs04	o7_ar5_1	st_e27	synthes3
deb7	ex1263a	fo8	no7_ar2_1	nvs05	o7	st_e29	tln2
deb8	ex1263	fo9_ar3_1	no7_ar25_1	nvs06	o8_ar4_1	st_e31	tln4
deb9	ex1264a	fo9_ar4_1	no7_ar3_1	nvs07	o9_ar4_1	st_e32	tln5
detf1	ex1264	fo9_ar5_1	no7_ar4_1	nvs08	oaer	st_e35	tln6
du-opt5	ex1265a	fo9	no7_ar5_1	nvs09	ortez	st_e36	tln7
du-opt	ex1265	fuel	nous1	nvs10	parallel	st_e38	tloss
eg_all_s	ex1266a	gastrans	nous2	nvs11	prob02	st_miqp1	tls2
eg_disc2_s	ex1266	gbd	nuclear14a	nvs12	prob03	st_miqp2	tls4
eg_disc_s	ex3	gear2	nuclear14b	nvs13	prob10	st_miqp3	tls5
elf	ex3pb	gear3	nuclear14	nvs14	procsel	st_miqp4	tltr
eniplac	ex4	gear4	nuclear24a	nvs15	product	st_miqp5	uselinear
enpro48	fac1	gear	nuclear24b	nvs16	qap	stockcycle	util
enpro48pb	fac2	gkocis	nuclear24	nvs17	qapw	st_test1	var_con10
enpro56	fac3	hmittelman	nuclear25a	nvs18	ravem	st_test2	var_con5
enpro56pb	feedtray2	johnall	nuclear25b	nvs19	ravempb	st_test3	water4
ex1221	feedtray	m3	nuclear25	nvs20	risk2bpb	st_test4	waterx
ex1222	fo7_2	m6	nuclearva	nvs21	saa_2	st_test5	waterz

solution (see Table 2.2). The instances for which the time limit is reached without finding

Table 2.2: Instances for which the feasible solution found is also the best-know solution

ex1222	nuclear24b	nuclearvd	st_e27
ex1266a	nuclear24	nuclearve	st_e32
feedtray2	nuclear25a	nuclearvf	st_miqp1
nuclear14a	nuclear25	nvs03	st_test1
nuclear14b	nuclearva	nvs15	st_test5
nuclear14	nuclearvb	prob02	tln2
nuclear24a	nuclearvc	prob03	tltr

Table 2.3: Instances for which no feasible solution was found within the time limit

deb10	fo9_ar25_1	nuclear49a	tln12
ex1252	gasnet	nuclear49b	tls12
fo8_ar25_1	lop97ic	nuclear49	tls6
fo8_ar3_1	lop97icx	product2	tls7
fo9_ar2_1	nuclear10a	space960	

any MINLP feasible solution are 19, see Table 2.3. The remaining 16 instances encounter some problems during the execution (see Table 2.4).

Table 2.4: Instances with problems during the execution

4stufen	ex1252a	risk2b	super3
beuster	nuclear104	st_e40	super3t
cecil_13	nuclear10b	super1	waste
eg_int_s	pump	super2	windfac

2.5 Conclusions

In this chapter we presented a Feasibility Pump (FP) algorithm aimed at solving non-convex Mixed Integer Non-Linear Programming problems. The proposed algorithm is tailored to limit the impact of the non-convexities in the MINLPs. These difficulties aare extensively discussed. In preliminary results we show the algorithm behaves well with general problems presenting computational results on instances taken from MINLPLib.

Chapter 3

A Global Optimization Method for a Class of Non-Convex MINLP Problems

[1]

3.1 Introduction

The global solution of practical instances of Mixed Integer Non-Linear Programming (MINLP) problems has been considered for some decades. Over a considerable period of time, technology for the global optimization of convex MINLP (i.e. the continuous relaxation of the problem is a convex program) had matured (see, for example, [45, 108, 20]), and rather recently there has been considerable success in the realm of global optimization of non-convex MINLP (see, for example, [111, 99, 84, 14]).

Global optimization algorithms, e.g., spatial Branch-and-Bound approaches like those implemented in codes like BARON [111] and Couenne [14], have had substantial success in tackling complicated, but generally small scale, non-convex MINLPs (i.e., mixed-integer nonlinear programs having non-convex continuous relaxations). Because they are aimed at a rather general class of problems, the possibility remains that larger instances from a simpler class may be amenable to a simpler approach.

We focus on separable MINLPs, that is where the objective and constraint functions are sums of univariate functions. There are many problems that are already in such a form, or can be brought into such a form via some simple substitutions. In fact, the first step in spatial Branch-and-Bound is to bring problems into nearly such a form. For our purposes, we shift that burden back to the modeler. We have developed a simple algorithm, implemented at the level of a modeling language (in our case AMPL, see [55]), to attack such separable problems. First, we identify subintervals of convexity and concavity for the univariate functions using external calls to MATLAB [91]. With such an identification at hand, we develop a convex MINLP relaxation of the problem (i.e., as a mixed-integer non-linear programs having a convex continuous relaxations). Our convex MINLP relaxation differs from those typically

[1]This is a working paper with Jon Lee and Andreas Wächter (Department of Mathematical Sciences, IBM T.J. Watson Research Center, Yorktown Heights, NY). This work was partially developed when the author of the thesis was visiting the IBM T.J. Watson Research Center and their support is gratefully acknowledged.

employed in spatial Branch-and-Bound; rather than relaxing the graph of a univariate function on an interval to an enclosing polygon, we work on each subinterval of convexity and concavity separately, using linear relaxation on only the "concave side" of each function on the subintervals. The subintervals are glued together using binary variables. Next, we employ ideas of spatial Branch-and-Bound, but rather than branching, we repeatedly refine our convex MINLP relaxation by modifying it at the modeling level. We attack our convex MINLP relaxation, to get lower bounds on the global minimum, using the code Bonmin [20, 26] as a black-box convex MINLP solver. Next, by fixing the integer variables in the original non-convex MINLP, and then locally solving the associated non-convex NLP relaxation, we get an upper bound on the global minimum, using the code Ipopt [123]. We use the solutions found by Bonmin and Ipopt to guide our choice of further refinements.

We implemented our framework using the modeling language AMPL. In order to obtain all of the information necessary for the execution of the algorithm, external software, specifically the tool for high-level computational analysis MATLAB, the convex MINLP solver Bonmin and the NLP solver Ipopt, are called directly from the AMPL environment. A detailed description of each part and of the entire algorithmic framework is provided in S3.2.

We present computational results in S3.3. Some of the instances used arise from specific applications; in particular, Uncapacitated Facility Location and also Hydro Unit Commitment and Scheduling. We also present computational results on selected instances of GLOBALLib and MINLPLib. We have had modest success in our preliminary computational experiments. In particular, we see very few major iterations occurring, with most of the time is spent in the solution of a small number of convex MINLPs. An advantage of our approach is that further advances in technology for convex MINLP will immediately give us a proportional benefit.

3.2 Our algorithmic framework

We focus now on separable non-convex MINLP problems. Without loss of generality, we take them to be of the form

$$\begin{array}{l} \min \sum_{j\in N} C_j x_j \\ \quad \text{subject to} \\ f(x) \le 0 \ ; \\ r_i(x) + g_i(x_{h(i)}) \le 0 \ , \ \forall i \in M \ ; \\ L_j \le x_j \le U_j \ , \ \forall j \in N \ ; \\ x_j \text{ integer}, \ \forall j \in I \ , \end{array} \tag{$\mathcal{P}$}$$

where $N := \{1, 2, \ldots, n\}$, $f : \mathbb{R}^n \to \mathbb{R}^p$ and $r_i : \mathbb{R}^n \to \mathbb{R}$, $\forall i \in M$, are convex functions, $h : M \to N$, the $g_i(x_{h(i)}) : \mathbb{R} \to \mathbb{R}$ are non-convex univariate function $\forall i \in M$, $H := \{h(i) \ : \ i \in M\} \subseteq N$ and $I \subseteq N$. We can take each L_j and U_j to be finite or infinite for $j \in N \setminus H$, but for $j \in H$ we assume that these are finite bounds.

Note that $g_i(x_{h(i)})$ can also be a piecewise-defined function, but each piece should be a continuous univariate function. Without loss of generality, we have taken the objective function as linear and all of the constraints to be inequalities, and further of the less-then-or-equal variety.

Our approach is an iterative technique based on three fundamental ingredients:

A reformulation method with which we obtain a convex MINLP relaxation $\mathcal{Q}$ of the original problem $\mathcal{P}$. Solving the convex MINLP relaxation $\mathcal{Q}$, we obtain a lower bound of our original problem $\mathcal{P}$;

A non-convex NLP restriction $\mathcal{R}$ of the original MINLP problem $\mathcal{P}$ obtained by fixing the variables within the set $\{x_j \ : \ j \in I\}$. Locally solving the non-convex NLP restriction $\mathcal{R}$, we obtain an upper bound of our original problem $\mathcal{P}$;

A refinement technique aimed at improving, at each iteration, the quality of the lower bound obtained by solving the convex MINLP relaxation $\mathcal{Q}$.

The main idea of our algorithmic framework is to iteratively solve a lower-bounding relaxation $\mathcal{Q}$ and an upper-bounding restriction $\mathcal{R}$ so that, in case the value of the UB and the LB are the same, the global optimality of the solution found is proven; otherwise we make a refinement to the lower-bounding relaxation $\mathcal{Q}$. At each iteration, we seek to decrease the gap between the lower and the upper bound, and hopefully, before too long, the gap will be within a tolerance value. In this case, or in the case a time/iteration limit is reached, the algorithm stops. If the gap is closed, we have found a global optimum, otherwise we have a heuristic solution (provided that the upper bound is not $+\infty$). The lower-bounding relaxation $\mathcal{Q}$ is a convex relaxation of the original non-convex MINLP problem, obtained by approximating the concave part of the non-convex univariate functions using piecewise linear approximation. The novelty in this part of the algorithmic framework is the new formulation of the convex relaxation: The function is approximated only where it is concave, and the convex parts of the functions are not approximated, but taken as they are. The convex relaxation proposed is described in details in Section 3.2.1. The upper-bounding restriction $\mathcal{R}$, described in Section 3.2.2, is obtained simply by fixing the variables with integrality constraints. The refinement technique consists of adding one or more breakpoints where needed, i.e. where the approximation of the non-convex function is bad and the solution of the lower-bounding problem lies. Different refinement strategies are described in Section 3.2.3, and in Section 3.3 computational experiments with the strategy that gives the best results are presented. Once the ingredients of the algorithmic framework are described in detail, we give a pseudo-code description of our algorithmic framework (see Section 3.2.4). We also discuss some considerations about the general framework and the similarities and differences with popular global optimization methods.

3.2.1 The lower-bounding convex MINLP relaxation $\mathcal{Q}$

To obtain our convex MINLP relaxation $\mathcal{Q}$ of the MINLP problem $\mathcal{P}$, we need to locate the subintervals of the domain of each univariate function g_i for which the function is uniformly convex or concave. For simplicity of notation, rather than refer to the constraint $r_i(x) + g_i(x_{h(i)}) \leq 0$, we consider a single constraint to have the form $r(x) + g(x_k) \leq 0$, where $r : \mathbb{R}^n \to \mathbb{R}$ is convex and $g : \mathbb{R} \to \mathbb{R}$ is a univariate non-convex function of x_k , for some k $(1 \leq k \leq n)$. We want to explicitly view each such g as a piecewise-defined function, where on each piece the function is either convex or concave. In practice, for each non-convex function g , we compute the points at which the convexity/concavity may change, i.e. the zeros of the second derivative of g , using `MATLAB`. In case a function g is naturally piecewise defined, we are essentially refining the piecewise definition of it in such a way that the convexity/concavity is uniform on each piece.

Consider the piecewise-defined univariate function

$$g(x_k) := \begin{cases} 1 + (x_k - 1)^3 \ , & \text{for } 0 \leq x_k \leq 2 \ ; \\ 1 + (x_k - 3)^2 \ , & \text{for } 2 \leq x_k \leq 4 \ , \end{cases}$$

depicted in FIG. 3.1. In addition to the breakpoints $x_k = 0, 2, 4$ of the definition of g, the convexity/concavity changes at $x_k = 1$, so by utilizing an additional breakpoint at $x_k = 1$ the convexity/concavity is now uniform on each piece.

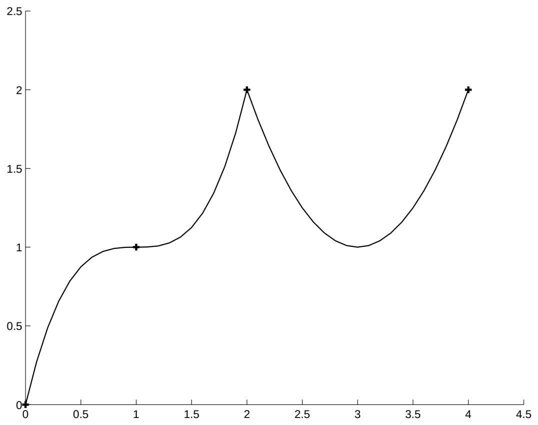


Figure 3.1: A piecewise-defined univariate function

Now, on each concave piece we can use a secant approximation to give now a piecewise-convex lower approximation of g .

EXAMPLE 1, continued. *Relative to* $g(x_k)$ *of* EXAMPLE 1, *we have the piecewise-convex lower approximation*

$$\underline{g}(x_k) := \begin{cases} x_k \ , & \textit{for } 0 \le x_k \le 1 \ ; \\ 1 + (x_k - 1)^3 \ , & \textit{for } 1 \le x_k \le 2 \ ; \\ 1 + (x_k - 3)^2 \ , & \textit{for } 2 \le x_k \le 4 \ , \end{cases}$$

depicted in FIG. 3.2.

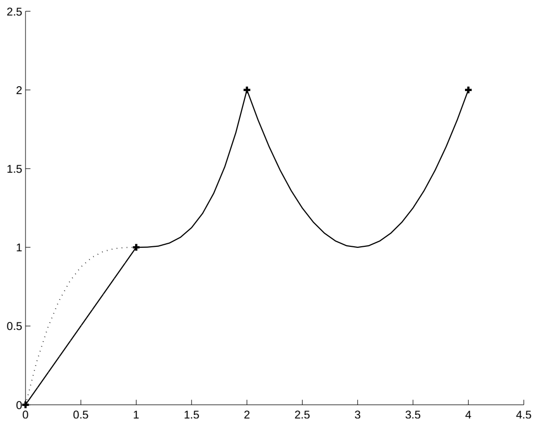


Figure 3.2: A piecewise-convex lower approximation

We can obtain a better lower bound by refining the piecewise-linear lower approximation on the concave pieces. We let

$$L_k =: P_0 < P_1 < \cdots < P_{\overline{p}} := U_k$$

be the ordered breakpoints at which the convexity/concavity of g changes, including, in the case of piecewise definition of g, the points at which the definition g changes. We define:

$[P_{p-1}, P_p]$:= the p-th subinterval of the domain of g ($p \in \{1 \ldots \overline{p}\}$);
$\check{H}$:= the set of indices of subintervals on which g is convex;
$\hat{H}$:= the set of indices of subintervals on which g is concave;

On the concave intervals, we will allow further breakpoints. We let B_p be the ordered set of breakpoints for the concave interval indexed by $p \in \hat{H}$. We denote these breakpoints as

$$P_{p-1} =: X_{p,1} < X_{p,2} < \cdots < X_{p,|B_p|} := P_p \ ,$$

and in our relaxation we will view g as lower bounded by the piecewise-linear function that has value $g(X_{p,j})$ at the breakpoints $X_{p,j}$, and is otherwise linear between these breakpoints.

EXAMPLE 1, continued again. *Utilizing further breakpoints, for example at $x_k = 1/3$ and $x_k = 2/3$, we can improve the piecewise-convex lower approximation to instead*

$$\underline{g}(x_k) := \begin{cases} \frac{19}{9}x_k \ , & \text{for } 0 \le x_k \le \frac{1}{3} \ ; \\ \frac{19}{27} + \frac{7}{9}\left(x_k - \frac{1}{3}\right) \ , & \text{for } \frac{1}{3} \le x_k \le \frac{2}{3} \ ; \\ \frac{26}{27} + \frac{1}{9}\left(x_k - \frac{2}{3}\right) \ , & \text{for } \frac{2}{3} \le x_k \le 1 \ ; \\ 1 + (x_k - 1)^3 \ , & \text{for } 1 \le x_k \le 2 \ ; \\ 1 + (x_k - 3)^2 \ , & \text{for } 2 \le x_k \le 4 \ , \end{cases}$$

depicted in FIG. 3.3.

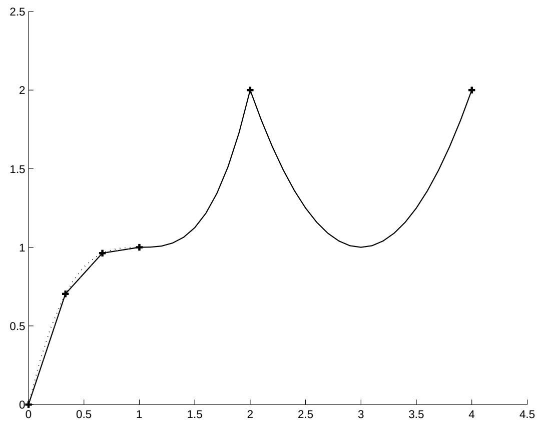


Figure 3.3: An improved piecewise-convex lower approximation

Next, we define further variables to manage our convexification of g on its domain.

z_p := a binary variable indicating if $x_k \geq P_p$ $(p = 1, \dots, \overline{p} - 1)$;

δ_p := a continuous variable assuming a positive value iff $x_k \geq P_{p-1}$ $(p = 1, \dots, \overline{p})$;

$\alpha_{p,b}$:= weight of breakpoint b in the piecewise-linear approximation of the interval indexed by p $(p \in \hat{H},\ b \in B_p)$.

In the convex relaxation of the original MINLP $\mathcal{P}$, we substitute a constraint of the form $r(x) + g(x_k) \leq 0$ with the following set of new constraints:

$$P_0 + \sum_{p=1}^{\overline{p}} \delta_p - x_k = 0 \ ; \tag{3.1}$$

$$\delta_p - (P_p - P_{p-1}) z_p \geq 0 \ , \ \forall p \in \check{H} \cup \hat{H} \ ; \tag{3.2}$$

$$\delta_p - (P_p - P_{p-1}) z_{p-1} \leq 0 \ , \ \forall p \in \check{H} \cup \hat{H} \ ; \tag{3.3}$$

$$P_{p-1} + \delta_p - \sum_{b \in B_p} X_{p,b}\, \alpha_{p,b} = 0 \ , \ \forall p \in \hat{H} \ ; \tag{3.4}$$

$$\sum_{b \in B_p} \alpha_{p,b} = 1 \ , \ \forall p \in \hat{H} \ ; \tag{3.5}$$

$$\{\alpha_{p,b} : b \in B_p\} := \text{SOS2} \ , \qquad \forall p \in \hat{H} \ ; \tag{3.6}$$

$$r(x) + \sum_{p \in \check{H}} g(P_{p-1} + \delta_p) + \sum_{p \in \hat{H}} \sum_{b \in B_p} g(X_{p,b})\, \alpha_{p,b} - \sum_{p=1}^{\overline{p}-1} g(P_p) \leq 0 \ , \tag{3.7}$$

with two dummy variables $z_0 := 1$ and $z_{\overline{p}} := 0$.

Constraints (3.1–3.3) together with the definition of the z variables ensure that, given an x_k value, say $x_k^* \in [P_{p^*-1}, P_{p^*}]$:

$$\delta_p = \begin{cases} P_p - P_{p-1} \ , & \text{if } 1 \leq p \leq p^* - 1 \ ; \\ x_k^* - P_{p-1} \ , & \text{if } p = p^* \ ; \\ 0 \ , & \text{otherwise.} \end{cases}$$

Constraints (3.4–3.6) ensure that, for each concave interval, the convex combination of the breakpoints is correctly computed. Finally, constraint (3.7) approximates the original non-convex constraint. Each single term of the first and the second summations, using the definition of δ_p , reduces, respectively, to

$$g(P_{p-1} + \delta_p) = \begin{cases} g(P_p) \ , & \text{if } p \in \{1, \dots, p^* - 1\} \ ; \\ g(x_k^*) \ , & \text{if } p = p^* \ ; \\ g(P_{p-1}) \ , & \text{if } p \in \{p^* + 1, \dots, \overline{p}\} \ , \end{cases}$$

and

$$\sum_{b \in B_p} g(X_{p,b})\, \alpha_{p,b} = \begin{cases} g(P_p) \ , & \text{if } p \in \{1, \dots, p^* - 1\} \ ; \\ \sum_{b \in B_{p^*}} g(X_{p^*,b})\, \alpha_{p^*,b} \ , & \text{if } p = p^* \ ; \\ g(P_{p-1}) \ , & \text{if } p \in \{p^* + 1, \dots, \overline{p}\} \ , \end{cases}$$

reducing constraint (3.7) to

$$r(x) + \sum_{p=1}^{p^*-1} g(P_p) + \gamma + \sum_{p=p^*+1}^{\overline{p}} g(P_{p-1}) - \sum_{p=1}^{\overline{p}-1} g(P_p) = r(x) + \gamma \le 0 \ ,$$

with

$$\gamma = \begin{cases} g(P_p - x_h^*) \ , & \text{if } p^* \in \check{H} \ ; \\ \sum_{b \in B_{p^*}} g(X_{p^*,b}) \ \alpha_{p^*b} \ , & \text{if } p^* \in \hat{H} \ . \end{cases}$$

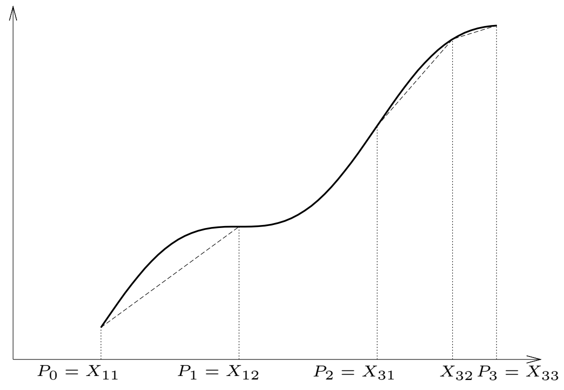


Figure 3.4: The convex relaxation

It is important to note that if we utilized a very large number of breakpoints at the start, solving the resulting convex MINLP $\mathcal{Q}$ would mean essentially solving globally the original MINLP $\mathcal{P}$. But of course such a convex MINLP $\mathcal{Q}$ would be too hard to be solved in practice. With our algorithmic framework, we dynamically seek a smaller convex MINLP $\mathcal{Q}$, thus generally more easily solvable, which we can use to guide the non-convex NLP restriction $\mathcal{R}$ to a good local solution, eventually settling on and proving global optimality of such a solution to the original MINLP $\mathcal{P}$.

3.2.2 The upper-bounding non-convex NLP restriction $\mathcal{R}$

Given a solution, typically an optimum $\underline{x}$ of the convex MINLP relaxation $\mathcal{Q}$, the upper-bounding restriction $\mathcal{R}$ is defined as the non-convex NLP:

$$\begin{array}{l} \min \ \sum_{j \in N} C_j x_j \\ \quad \text{subject to} \\ f(x) \le 0 \ ; \\ r_i(x) + g_i(x_{h(i)}) \le 0 \ , \ \forall i \in M \ ; \\ L_j \le x_j \le U_j \ , \ \forall j \in N \ ; \\ x_j = \underline{x}_j, \ \forall j \in I \ . \end{array} \qquad (\mathcal{R})$$

A solution of this non-convex NLP $\mathcal{R}$ is a heuristic solution of the non-convex MINLP problem $\mathcal{P}$ for two reasons: (i) the integer variables x_j , $j \in I$, might not be fixed to globally optimal values; (ii) the NLP $\mathcal{R}$ is non-convex, and so even if the integer variables x_j , $j \in I$, are fixed to globally optimal values, the NLP solver may well only find a local optimum of the non-convex NLP $\mathcal{R}$. This consideration emphasizes the importance of the lower-bounding relaxation $\mathcal{Q}$ for the guarantee of the global optimality. The upper-bounding problem resolution could be seen as a "verification phase" in which a solution of the convex MINLP relaxation $\mathcal{Q}$ is tested to be really feasible for the non-convex MINLP $\mathcal{P}$.

3.2.3 The refinement technique

At the end of each iteration, we have two solutions: $\underline{x}$, the solution of the lower-bounding convex MINLP relaxation $\mathcal{Q}$, and $\overline{x}$, the solution of the upper-bounding non-convex NLP restriction $\mathcal{R}$. If $\sum_{j\in N} C_j \underline{x}_j < \sum_{j \in N} C_j \overline{x}_j$ within a certain tolerance, the solution $\underline{x}$ is not feasible for the original MINLP problem $\mathcal{P}$. In order to be able to continue, we want to refine the approximation of the lower-bounding convex MINLP relaxation $\mathcal{Q}$ by adding further breakpoints. There are different points at which we can add further breakpoints, and we tried several strategies including:

For each $i \in M$, add a breakpoint to g_i where the lower-bounding solution is: If $\underline{x}_{h(i)}$ lies on a concave interval of g_i , and it is not (very) close to any breakpoint already defined for g_i ;

For each $i \in M$, add a breakpoint to g_i where the upper-bounding solution is: If $\overline{x}_{h(i)}$ lies on a concave interval of g_i , and it is not (very) close to any breakpoint already defined for g_i ;

Use both the strategies above;

For each $i \in M$, add a breakpoint to g_i where a combination of the two solutions lies: Add a breakpoint at $\lambda\ \underline{x} + (1-\lambda)\overline{x}$, for some $0 \leq \lambda \leq 1$, chosen so that the new breakpoint is not to close to any breakpoint already defined for g_i ;

It is also possible to add several breakpoints instead of just one, using strategies like those above. The strategy that we settled upon for our computational results is the third: Namely, we add a breakpoint where $\underline{x}$ lies in order to converge and one where $\overline{x}$ lies to speed up the convergence.

3.2.4 The algorithmic framework

Algorithm 3 details our algorithmic framework, while Figure 3.5 depicts it at a high level.

At each iteration, the lower-bounding MINLP relaxation $\mathcal{Q}$ and the upper-bounding NLP restriction $\mathcal{R}$ are redefined: What changes in $\mathcal{Q}$ are the sets of breakpoints that approximate the concave intervals of the non-convex functions. At each iteration, the number of breakpoint used increases, and so does the accuracy of the approximation. What may change in $\mathcal{R}$ are the values of the fixed integer variables $\underline{x}_j$, $j \in I$. Moreover, what changes is the starting point given to the NLP solver, derived from an optimal solution of the lower-bounding MINLP relaxation $\mathcal{Q}$.

Our algorithmic framework bears comparison with spatial Branch-and-Bound, a successful technique in global optimization. In particular:

Algorithm 3 The algorithmic framework

```
$LB := -\infty$; $UB := +\infty$;
Find $P_p^i$ , $\hat{H}^i$, $\check{H}^i$, $X_{pb}^i$ $(\forall i \in M, p \in \{1 \dots \overline{p}^i\}, b \in B_p^i)$.
repeat
  Solve the convex MINLP relaxation $\mathcal{Q}$ of the original problem $\mathcal{P}$ ;
  if $(\mathrm{val}(\mathcal{Q}) > LB)$ then
    $LB := \mathrm{val}(\mathcal{Q})$;
  end if
  Solve the non-convex NLP restriction $\mathcal{R}$ of the original problem $\mathcal{P}$ ;
  if $(\mathrm{val}(\mathcal{R}) < UB)$ then
    $UB := \mathrm{val}(\mathcal{R})$;
  end if
  if $(UB - LB \geq \Delta)$ then
    Update $B_p^i$ , $X_{pb}^i$ ;
  end if
until $((UB - LB \leq \Delta)$ or (alternative termination criterion))
return the solution $\overline{x}$ of the non-convex NLP restriction $\mathcal{R}$ ;
```

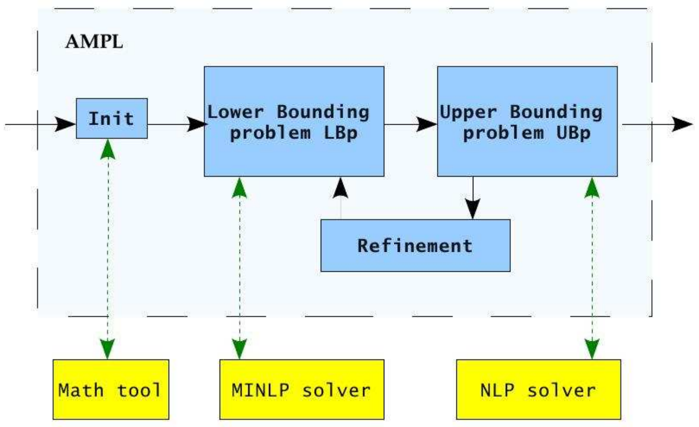


Figure 3.5: The algorithmic framework

during the refining phase, the parts in which the approximation is bad are discovered and the approximation is improved there, but we do it by adding one or more breakpoints instead of branching on a continuous variable as in spatial Branch-and-Bound;

unlike spatial Branch-and-Bound, our approach does not utilize an expression tree; it works directly on the broad class of separable non-convex MINLPs of the form $\mathcal{P}$, and of course problems that can be put in such a form;

unlike spatial Branch-and-Bound, our method can be effectively implemented at the modeling-language level.

3.3 Computational results

We implemented our algorithmic framework as an AMPL script, and we used MATLAB as a tool for numerical analysis, Bonmin as our convex MINLP solver, and Ipopt as our NLP solver. In this section we present computational results for three problem categories. The tests were executed by sequentially running the code on a single processor of an Intel Core2 CPU 6600, 2.40 GHz, 1.94 GB of RAM using a time limit of 2 hours.

For each set of problems, the non-convex MINLP model $\mathcal{P}$ is presented, and a reference to a paper in which the problem is described in more detail is given. A table with computational results exhibits the behavior of our algorithm on some instances of each problem. Each table presents ten columns:

the instance name;

the number of iteration the data are referred to;

the value of the lower bound;

the value of the upper bound;

if the integer variables are different with respect to ones used in the previous iteration (the values fixed in the UB problem, if applicable);

the CPU time needed to solve the convex relaxation to optimality (in seconds);

the number of breakpoints added at the previous iteration;

the number of variables/integers/constraints;

the lower and the upper bound computed by Couenne. If the gap is closed within the time limit, the LB column reports the CPU running time and the UB column reports the value of the global solution found.

3.3.1 Uncapacitated Facility Location (UFL) problem

The UFL application is presented in [68]. (Note that here the set of customers is denoted with T and the set of facilities is denoted with K (w_{kt} is the fraction of demand of customer

t satisfied by facility k for each $t \in T, k \in K$).) We report the mathematical model used for the computational results of Table 3.1:

$$
\begin{array}{l}
\min \ \sum_{k \in K} C_k y_k + \sum_{t \in T} v_t \\
\quad \text{subject to} \\
v_t \geq - \sum_{k \in K} S_{kt} s_{kt} \ , \ \forall \ t \in T \ ; \\
s_{kt} \leq g_{kt}(w_{kt}) \ , \ \forall \ t \in T \ ; \\
w_{kt} \leq y_k \ , \ \forall \ t \in T, k \in K \ ; \\
\sum_{k \in K} w_{kt} = 1 \ , \ \forall \ t \in T \ ; \\
w_{kt} \geq 0 \ , \ \forall \ t \in T, k \in K \ ; \\
y_k \in \{0, 1\} \ , \ \forall \ k \in K \ .
\end{array}
$$

Figure 3.6 depicts the three different non-linear function $g_{kt}(w_{kt})$ were used for computational results presented in Table 3.1. The piecewise linear functions are the segments linking the lower and the upper bound and the point were the curvature changes. Table 3.1 shows good

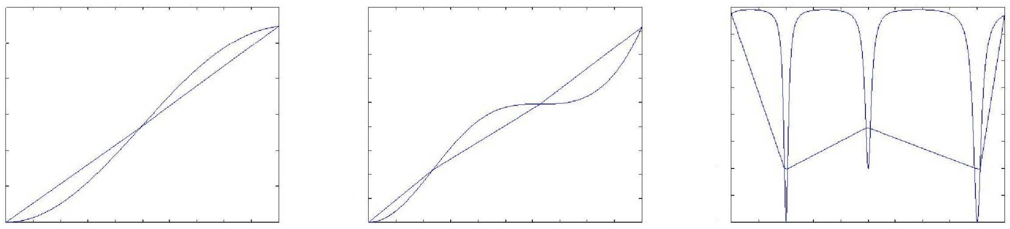


Figure 3.6: UFL: how $-g_{kt}(w_{kt})$ looks like in the three instances.

Table 3.1: Results for Uncapacitated Facility Location problem

inst	cycle	LB	UB	int change	t MINLP	br added	(var;int;constr)	Couenne LB	Couenne UB
ufl_1	1	4,122.00	4,330.40	-	1.41	-	(153;39;228)	2,315.08”	4,330.40
	2	4,324.78	4,330.40	no	12.34	11			
	3	4,327.72	4,330.40	no	23.20	5			
	4	4,328.99	4,330.40	no	38.36	5			
	5	4,330.04	4,330.40	no	61.03	5			
	6	4,330.24	4,330.40	no	94.95	5			
	7	4,330.36	4,330.40	no	131.59	5			
	8	4,330.38	4,330.40	no	181.14	5			
	9	4,330.39	4,330.40	no	249.04	5			
ufl_2	1	27,516.60	27,516.60	-	3.93	-	(189;57;264)	410.47”	27,516.60
ufl_3	1	1,947.88	2,756.89	-	3.88	-	(79;21;101)	0.76”	2,292.78
	2	2,064.26	2,756.89	no	3.90	2			
	3	2,292.74	2,292.78	no	4.17	2			

performance of the proposed algorithm. In particular, instance `ufl_1` is solved in less than 800 seconds with respect to 2,315.08 seconds needed by Couenne, instance `ufl_2` in less than 4 seconds with respect to 410.47 seconds. In instance `ufl_3` Couenne performs better than the proposed algorithm, 0.76 with respect to 13.

3.3.2 Hydro Unit Commitment and Scheduling problem

The Hydro Unit Commitment and Scheduling problem is described in see Chapter 6. (Note that in this case the index j has a different meaning with respect to the model presented in

Section 3.1: here it represents the turbine/pump unit.) In the following the model used for the computational results of Table 3.2:

$$
\begin{array}{l}
\min\ -\sum_{j\in J}\sum_{t\in T}\Big(\Delta t\,\Pi_t\,p_{jt} - C_j\,\widetilde{w}_{jt} - (D_j + \Pi_t E_j)\widetilde{y}_{jt}\Big) \\
\quad \text{subject to} \\
v_{\overline{t}} - V_{\overline{t}} = 0 \ ; \\
v_t - v_{t-1} - 3600\,\Delta t\,(I_t - \sum_{j\in J} q_{jt} - s_t) = 0 \ , \ \forall t \in T \ ; \\
q_{jt} - (Q_j^-\ u_{jt} + \underline{Q}_j\ g_{jt}) \geq 0 \ , \ \forall j \in J, t \in T \ ; \\
q_{jt} - (Q_j^-\ u_{jt} + \overline{Q}_j\ g_{jt}) \leq 0 \ , \ \forall j \in J, t \in T \ ; \\
\sum_{j\in J}(q_{jt} - q_{j(t-1)}) + \Delta q^- \geq 0 \ , \ \forall t \in T \ ; \\
\sum_{j\in J}(q_{jt} - q_{j(t-1)}) - \Delta q^+ \leq 0 \ , \ \forall t \in T \ ; \\
s_t - \sum_{j\in J}(W_j\,\widetilde{w}_{jt} + Y_j\,\widetilde{y}_{jt}) \geq 0 \ , \ \forall t \in T \ ; \\
\sum_{j\in J} q_{jt} + s_t - \underline{\Theta} \geq 0 \ , \ \forall t \in T \ ; \\
g_{jt} - g_{j(t-1)} - (\widetilde{w}_{jt} - w_{jt}) = 0 \ , \ \forall j \in J, t \in T \ ; \\
\widetilde{w}_{jt} + w_{jt} \leq 1 \ , \ \forall j \in J, t \in T \ ; \\
u_{jt} - u_{j(t-1)} - (\widetilde{y}_{jt} - y_{jt}) = 0 \ , \ \forall j \in J, t \in T \ ; \\
\widetilde{y}_{jt} + y_{jt} \leq 1 \ , \ \forall j \in J, t \in T \ ; \\
g_{jt} + u_{kt} \leq 1 \ , \ \forall j, k \in J, t \in T \ ; \\
\sum_{j\in J} u_{jt} \leq \overline{n} - 1 \ , \ \forall t \in T \ ; \\
p_{jt} - \varphi(q_{jt}) = 0 \ , \ \forall j \in J, t \in T.
\end{array}
$$

In Figure 3.7, the plot of three different non-linear functions $\varphi(q_{jt})$ which were used is depicted. The piecewise linear functions are the segments linking the lower and the upper bound and the point were the curvature changes. Also Table 3.2 shows a good performance of the

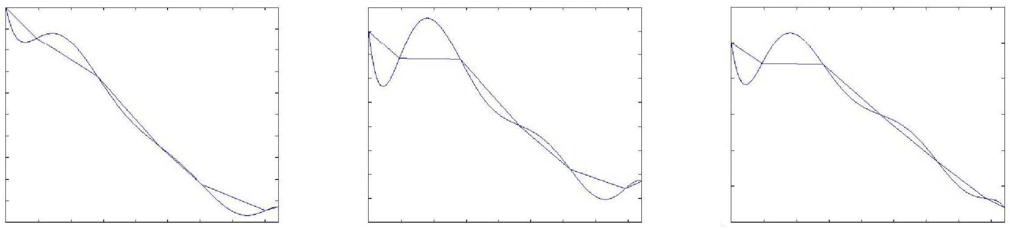


Figure 3.7: Hydro UC: how $-\varphi(q_{jt})$ looks like in the three instances

Table 3.2: Results for Hydro Unit Commitment and Scheduling problem

inst	cycle	LB	UB	int change	t MINLP	br added	(var;int;constr)	Couenne LB	Couenne UB
hydro_1	1	-10,231.00	-10,140.80	-	16.37	-	(324;142;445)	-11,229.80	-10,140.80
	2	-10,140.80	-10,140.80	no	22.99	4			
hydro_2	1	-3,950.70	-3,891.22	-	31.30	-	(324;142;445)	-12,104.40	-2,910.91
	2	-3,950.58	-3,891.22	no	34.51	2			
	3	-3,950.58	-3,891.22	no	38.54	2			
	4	-3,932.18	-3,932.18	no	54.89	2			
hydro_3	1	-4,753.85	-4,634.40	-	147.37	-	(324;142;445)	-12,104.40	-3,703.07
	2	-4,719.93	-4,660.19	no	264.22	4			
	3	-4,710.74	-4,710.73	yes	339.49	2			

proposed algorithm. It is able to find the global optimum of the three instances within the time limit, but Couenne does not solve to global optimality any of the instances.

3.3.3 GLOBALLib and MINLPLib instances

We selected 13 instances from GLOBALLib `http://www.gamsworld.org/global/globallib.htm` and 9 from MINLPLib `http://www.gamsworld.org/minlp/minlplib.htm` to test our algorithm. Because our algorithm is suitable only for instances with univariate non-linear functions, we reformulated them in order to be able to apply our technique to the reformulated problem. Of course, this is not possible for every instance of the libraries mentioned, so we selected only some instances. In fact, we also had to exclude other instances for another issue: we need the independent variables $x_{h(i)}$ involved in the univariate non-linear constraints to be bounded $\forall i \in N$. We need the upper and the lower bound for these variables because we use them to compute the curvature changes within the MATLAB environment (see Section 3.2.1). We report in the first part of the table the GLOBALLib results. We ran the instances until the global optimum is found and proved. The second part of the table reports the MINLPLib results. The instances were selected among the ones reported in the computational results of a recent paper [14]. The tests are performed in the same machine. In Table 3.3 the third last reports the number of variables, integer variables and constraints of the original problem and the reformulated one. Table 3.3 shows that Couenne performs much better on GlobalLib instances. This can be probably explained by the fact that for small instances Couenne behaves very well in general. Moreover, the reformulation needed by the proposed algorithm to make these problems separable partially influence the performance making the model larger. Concerning the MinlpLib instances, 4 over 9 instances are solved to global optimality by both our algorithm and Couenne and Couenne performs better in 2 of these 4 instances. In the other 5 instances, the lower bound given by the proposed algorithm is always better (higher) than the one provided by Couenne. This result emphasizes the quality of the lower bound computed by the solution of the convex MINLP relaxation $\mathcal{Q}$. However, the upper bound computed by Couenne is better in 3 instances over 5.

3.4 Conclusions

In this Chapter, we proposed an algorithms for solving to global optimality separable MINLPs. Our simple algorithm, implemented within the AMPL modeling language, consists of a lower-bounding and an upper-bounding problem. For the definition of the lower-bounding problem, we identify subintervals of convexity and concavity for the univariate functions using external calls to MATLAB then we develop a convex MINLP relaxation of the problem approximating the concave intervals of each non-convex function with linear relaxation. The subintervals are glued together using binary variables. We iteratively refine our convex MINLP relaxation by modifying it at the modeling level. The upper-bounding problem is obtained by fixing the integer variables in the original non-convex MINLP, then locally solving the associated non-convex NLP relaxation. We presented preliminary computational experiments on real-world applications like Uncapacitated Facility Location and Hydro Plants Unit Commitment and Scheduling problems and instances of GlobalLib and MinlpLib. We compared our algorithm with the open-source solver Couenne obtaining a modest success.

An advantage of our approach is that further advances in technology for convex MINLP will immediately give us a benefit.

Table 3.3: Results for GLOBALLib and MINLPLib

inst	cycle	LB	UB	int change	t MINLP	br added	(var;int;constr)	Couenne LB	Couenne UB
ex14_2_1	1	0	0		9.96	-	(5;0;7) (122;0;6)	0”	0
ex14_2_2	1	0	0		5.61	-	(4;0;5) (56;0;4)	0.01”	0
ex14_2_6	1	0	0		13.41	-	(5;0;7) (164;0;6)	0.01”	0
ex14_2_7	1	0	0		12.03	-	(6;0;9) (277;0;8)	0”	0
ex2_1_1	1	-18.3182	-16.5	-	0.00	-	(5;0;1) (12;0;1)	0.05”	-17
	2	-18.2143	-16.5	-	0.06	1			
	3	-18	-16.5	-	0.12	1			
	4	-17.625	-17	-	0.20	1			
	5	-17	-17	-	0.32	2			
ex2_1_2	1	-213	-213	-	0.00	-	(6;0;2) (14;0;2)	0”	-213
ex2_1_3	1	-15	-15	-	0.00	-	(13;0;6) (24;0;6)	0”	-15
ex2_1_4	1	-11	-11	-	0.00	-	(6;0;4) (12;0;4)	0”	-11
ex2_1_5	1	-269.45	-268.015	-	0.00	-	(10;0;11) (29;0;11)	0.21”	-268.015
	2	-268.015	-268.015	-	0.22	2			
ex2_1_6	1	-44.40	-29.40	-	0.01	-	(10;0;5) (26;0;5)	0.04”	-39
	2	-40.50	-39.00	-	0.16	2			
	3	-40.16	-39.00	-	0.25	1			
	4	-39.00	-39.00	-	0.52	2			
ex9_2_2	1	55.56	100.00	-	0.00	-	(10;0;9) (38;0;5)	0.11”	100
	2	92.01	100.00	-	1.54	19			
	3	97.94	100.00	-	4.32	10			
	4	99.48	100.00	-	19.29	11			
	5	99.86	100.00	-	60.80	10			
	6	99.96	100.00	-	231.79	10			
ex9_2_3	1	-30.00	0.00	-	0.00	-	(16;0;15) (54;0;7)	0.08”	0
	2	-30.00	0.00	-	9.76	22			
	3	-27.16	0.00	-	17.27	10			
	4	-23.08	0.00	-	12.98	11			
	5	-20.56	0.00	-	34.75	9			
	6	-18.12	0.00	-	91.05	8			
	7	-12.05	0.00	-	297.97	14			
	8	-4.43	0.00	-	1,088.67	13			
	9	-4.21	0.00	-	3,102.74	6			
	10	-0.72	0.00	-	7,158.79	6			
	11	0.00	0.00	–	2,885.65	12			
ex9_2_6	1	-1.50	-1.00	-	0.01	-	(16;0;12) (57;0;6)	0.15”	-1
	2	-1.50	-1.00	-	7.88	29			
	3	-1.50	-1.00	-	7.90	25			
	4	-1.50	-1.00	-	28.47	19			
	5	-1.50	-1.00	-	34.63	15			
	6	-1.50	-1.00	-	127.61	18			
	7	-1.50	-1.00	-	356.12	17			
	8	-1.50	-1.00	-	354.22	13			
	9	-1.39	-1.00	-	411.72	12			
	10	-1.50	-1.00	-	682.30	19			
	11	-1.50	-1.00	-	444.82	14			
	12	-1.38	-1.00	-	981.41	13			
	13	-1.00	-1.00	-	1,311.27	21			
du-opt	1	3.556	3.556	-	4.10	-	(20;13;8) (242;18;230)	51.98”	3.556
du-opt5	1	8.073	8.073	-	7.13	-	(18;11;6) (239;15;227)	29.83”	8.073
fo7	1	(8.759)	22.518	-	7,200.00	-	(112;42;211) (338;42;437)	1.885	23.218
m6	1	82.256	82.256	-	175.07	-	(84;30;157) (254;30;327)	91.73”	82.256
no7_ar2_1	1	(90.583)	127.774	-	7,200.00	-	(112;42;269) (394;41;551)	74.63	111.145
no7_ar3_1	1	(81.5393)	107.869	-	7,200.00	-	(112;42;269) (394;41;551)	47.45	112.032
no7_ar4_1	1	(76.402)	104.534	-	7,200.00	-	(112;42;269) (394;41;551)	43.23	98.884
o7_2	1	(79.365)	124.324	-	7,200.00	-	(112;42;211) (338;42;437)	7.654	128.085
stockcycle	1	119,949	119,949	-	188.26	-	(480;432;97)(578;480;195)	63.93”	119,949

Chapter 4

A Method for Approximating Non-Linear Functions of Two Variables

[1]

4.1 Introduction

In recent years, the increased efficiency of Mixed Integer Linear Programming (MILP) software tools has encouraged their use also in the solution of non-linear problems, bringing to the need for efficient techniques to linearize non-linear functions of one or more variables. The standard methodologies consist in the piecewise linear approximation of such functions.

For functions of a single variable, say, $f(x)$, the classical approach consists in introducing a number n of sampling coordinates $x_1, \dots, x_n$ on the x axis (*breakpoints*) on which the function is evaluated, with x_1 and x_n coinciding with the left and right extremes of the domain of x (see Figure 4.1(a)). For any given x value, say, $\overline{x}$, with $x_i \le \overline{x} \le x_{i+1}$, the function value is approximated by convex combination of $f(x_i)$ and $f(x_{i+1})$. Let λ be the (unique) value in $[0, 1]$ such that:

$$\overline{x} = \lambda x_i + (1 - \lambda) x_{i+1}. \tag{4.1}$$

Then the approximated value is:

$$f^a(\overline{x}) = \lambda f(x_i) + (1 - \lambda) f(x_{i+1}). \tag{4.2}$$

This methodology can alternatively be described through the slope $(f(x_{i+1}) - f(x_i))/(x_{i+1} - x_i)$ of the interpolating function, namely:

$$f^a(\overline{x}) = f(x_i) + (\overline{x} - x_i)\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \tag{4.3}$$

(from which one has $\lambda = (x_{i+1} - \overline{x})/(x_{i+1} - x_i)$).

[1]The results of this chapter appears in: C. D'Ambrosio, A. Lodi, S. Martello, Piecewise linear approximation of functions of two variables in MILP models, Technical Report OR-09-3, University of Bologna.

In order to use the above technique in a MILP solver it is necessary to include in the model variables and constraints that force any x value to be associated with the proper pair of consecutive breakpoints (or with a single one, in case $x \in \{x_1, \dots, x_n\}$). Let us introduce a continuous variable α_i for each breakpoint i, such that $\alpha_i \in [0, 1]$ $(i = 1, \dots, n)$. Let h_i be a binary variable associated with the ith interval $[x_i, x_{i+1}]$ $(i = 1, \dots, n-1)$, with dummy values $h_0 = h_n = 0$. The approximate value f^a can be then obtained by imposing the following constraints:

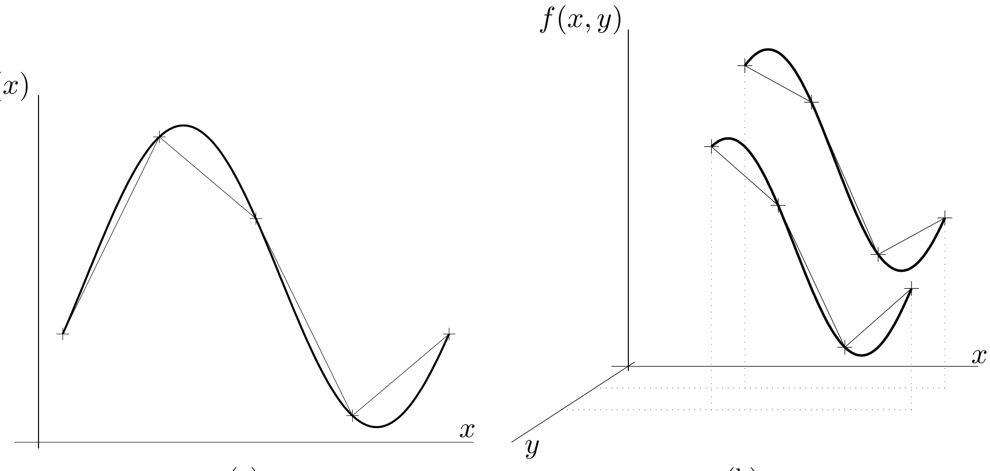


Figure 4.1: Piecewise linear approximation of a univariate function, and its adaptation to a function of two variables.

$$\begin{aligned}
\sum_{i=1}^{n-1} h_i &= 1 && (4.4)\\
\alpha_i &\le h_{i-1} + h_i \quad (i = 1, \dots, n) && (4.5)\\
\sum_{i=1}^{n} \alpha_i &= 1 && (4.6)\\
x &= \sum_{i=1}^{n} \alpha_i x_i && (4.7)\\
f^a &= \sum_{i=1}^{n} \alpha_i f(x_i). && (4.8)
\end{aligned}$$

Constraint (4.4) imposes that only one h_i, say, $h_{\bar{\imath}}$, takes the value 1. Hence constraints (4.5) impose that the only α_i values different from 0 can be $\alpha_{\bar{\imath}}$ and $\alpha_{\bar{\imath}+1}$. It follows from (4.6) and (4.7) that $\alpha_{\bar{\imath}} = \lambda$ and $\alpha_{\bar{\imath}+1} = 1 - \lambda$ (see (4.1)). Constraint (4.8) ensure then the correct computation of the approximate value according to (4.2).

In contexts of this type, the MILP constraints can be simplified by the so-called special ordered sets, introduced by Beale and Tomlin [13], and extensively studied by Lee and Wilson [80], Keha, de Farias and Nemhauser [75] and Martin, Moller and Moritz [89]. By defining a set of variables to be a *Special Ordered Set of type k* (SOSk), one imposes that at most k such variables can take a non-zero value, and that they must be consecutive. Most modern MILP solvers are capable of automatically handling special ordered sets of type 1 and 2. In our case, by defining the α variables to be a SOS2, one does not need to explicitly state h variables, so constraints (4.6)-(4.8) produce the correct computation. The additional advantage of this technique is that the enumerative phase may be enhanced by the internal use of special purpose branching rules.

The remainder of this chapter concentrates on the piecewise linear approximation of functions $f(x, y)$ of two variables. In Section 4.2 we present three approaches, and give a detailed description of how they can be embedded in a MILP model. The simplest method (Section 4.2.1) consists of using the one-variable technique above for a discretized set of y values. A more complex approach (Section 4.2.2) is based on the definition of triangles in the three-dimensional space, and can be seen as the extension of the one-variable technique (see Tomlin [120], Babayev [10], Lee and Wilson [80], Martin, Moller and Moritz [89] and Vielma and Nemhauser [122]). In Section 4.2.3 we give a full description of a third approach, recently used within an applied context (see Borghetti, D'Ambrosio, Lodi and Martello [28]), which appears particularly suitable for MILP modeling. In Section 4.3 we show that the three approaches do not dominate each other, and discuss advantages and drawbacks on the basis of some numerical examples. A detailed comparison within a MILP approach is reported in Section 4.3.2 with respect to an application in electric power generation.

4.2 The methods

In this section we describe three techniques for the piecewise linear approximation of functions of two variables.

4.2.1 One-dimensional method

An immediate adaptation of the one-variable technique to the case of functions of two variables is as follows. Let us introduce a number m of coordinates on the y axis, $y_1, \ldots, y_m$ (y_1 and y_m being the left and right extremes of the domain of y). For the jth interval $[y_j, y_{j+1})$, let $\widetilde{y}_j$ be the associated sampling coordinate (often the central point of the interval), leading to $m-1$ univariate functions $f(x, \widetilde{y}_j)$ ($j = 1, \ldots, m-1$). For any given y value, say, $\overline{y} \in [y_j, y_{j+1})$, the approximated function values $f^a(x, \overline{y})$ are then given by the piecewise linear approximation of $f(x, \widetilde{y}_j)$ with breakpoints $x_1, \ldots, x_n$ (see Figure 4.1(b)).

Let $\beta_1, \ldots, \beta_{m-1}$ be binary variables, defined as an SOS1, with β_j taking the value 1 if and only if the given value $\overline{y}$ belongs to $[y_j, y_{j+1})$. The approximate value f^a is then obtained through (4.6)-(4.7) and:

$$
\begin{aligned}
y &\leq \sum_{j=1}^{m-1} \beta_j y_{j+1} && (4.9)\\
y &\geq \sum_{j=1}^{m-1} \beta_j y_j && (4.10)\\
\sum_{j=1}^{m-1} \beta_j &= 1 && (4.11)\\
f^a &\leq \sum_{i=1}^{n} \alpha_i f(x_i, \widetilde{y}_j) + M(1-\beta_j) \quad (j = 1, \ldots, m-1) && (4.12)\\
f^a &\geq \sum_{i=1}^{n} \alpha_i f(x_i, \widetilde{y}_j) - M(1-\beta_j) \quad (j = 1, \ldots, m-1), && (4.13)
\end{aligned}
$$

where α is the SOS2 introduced in the previous section and M is a very large value ("big-M"). Constraints (4.9)-(4.11) impose $\beta_{\overline{\jmath}} = 1$ and $\beta_j = 0$ for $j \neq \overline{\jmath}$, $\overline{\jmath}$ being the interval which contains y. Constraints (4.12)-(4.13) are inactive if $\beta_j = 0$, hence providing $f^a = \sum_{i=1}^{n} \alpha_i f(x_i, \widetilde{y}_{\overline{\jmath}})$ for the correct interval $\overline{\jmath}$.

4.2.2 Triangle method

A more complex method can be obtained by extending the one-variable technique to the two-variable case. Consider again n sampling coordinates $x_1, \ldots, x_n$ on the x axis and m sampling coordinates $y_1, \ldots, y_m$ on the y axis, with x_1 and x_n (resp. y_1 and y_m) coinciding with the left and right extremes of the x (resp. y) domain. The function $f(x, y)$ is evaluated for each breakpoint (x_i, y_j) $(i = 1, \ldots, n; j = 1, \ldots, m)$.

For any given (x, y) point, say, $(\overline{x}, \overline{y})$, with $x_i \leq \overline{x} \leq x_{i+1}$ and $y_j \leq \overline{y} \leq y_{j+1}$, let us consider the rectangle (see Figure 4.2(a)) of vertices (x_i, y_j), (x_{i+1}, y_j), (x_{i+1}, y_{j+1}), (x_i, y_{j+1}), and the two triangles produced by its diagonal $[(x_i, y_j)(x_{i+1}, y_{j+1})]$. (The triangles produced by the other diagonal could equivalently be used.) The function value is approximated by convex combination of the function values evaluated at the vertices of the triangle containing $(\overline{x}, \overline{y})$. Namely (see Figure 4.2(b)),

$$
f^a(\overline{x}, \overline{y}) = \lambda f(x_i, y_j) + \mu f(x_{i+1}, y_{j+1}) + (1 - \lambda - \mu)\overline{f}, \tag{4.14}
$$

where

$$
\overline{f} = \begin{cases} f(x_{i+1}, y_j) & \text{if } \overline{y} \leq y_j + (\overline{x} - x_i)(y_{j+1} - y_j)/(x_{i+1} - x_i) \\ f(x_i, y_{j+1}) & \text{otherwise} \end{cases} \tag{4.15}
$$

and $\lambda \in [0, 1]$, $\mu \in [0, 1]$ and $(1-\lambda-\mu) \in [0, 1]$ are the weights of the convex combination of the vertices of the appropriate triangle which contains $(\overline{x}, \overline{y})$. In a MILP model this is obtained by introducing $n\,m$ continuous variables $\alpha_{ij} \in [0, 1]$ (one per breakpoint) and computing the

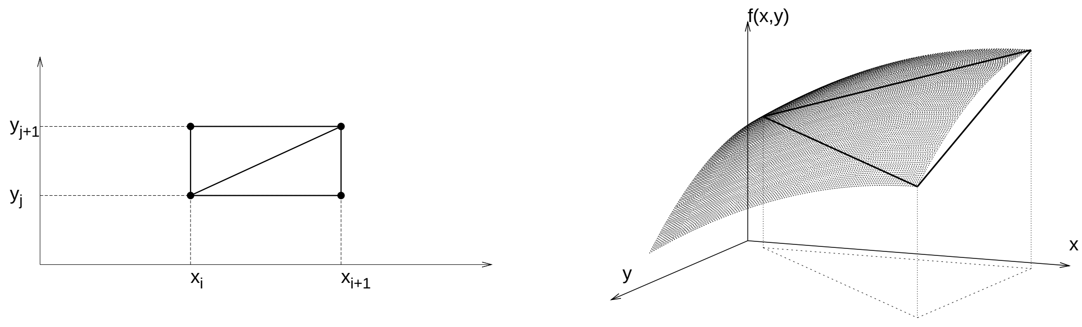


Figure 4.2: Geometric representation of the triangle method.

convex combinations by extending (4.6)-(4.8) to the three-dimensional space as follows:

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{ij} = 1 \tag{4.16}$$

$$x = \sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{ij}x_i \tag{4.17}$$

$$y = \sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{ij}y_j \tag{4.18}$$

$$f^a = \sum_{i=1}^{n}\sum_{j=1}^{m}\alpha_{ij}f(x_i,y_j). \tag{4.19}$$

Variables λ_{ij} should be defined as a specific SOS3. However, differently from what happens for SOS1 and SOS2, current MILP solvers do not have an automatic syntax to impose an SOS3, so, for the sake of completeness, we give here the analogue of constraints (4.4)-(4.5), to be added to the above model. Consider the rectangle corresponding to intervals $[x_i, x_{i+1})$ and $[y_j, y_{j+1})$: we associate binary variables h^u_{ij} and h^l_{ij} respectively to the upper and lower triangle in the rectangle (see Figure 4.2(a)), with dummy values $h^*_{0*} = h^*_{*0} = h^*_{n*} = h^*_{*m} = 0$ at the extremes. The additional constraints are then:

$$\sum_{i=1}^{n-1}\sum_{j=1}^{m-1}(h^u_{ij}+h^l_{ij}) = 1 \tag{4.20}$$

$$\alpha_{ij} \leq h^u_{ij}+h^l_{ij}+h^u_{i,j-1}+h^l_{i-1,j-1}+h^u_{i-1,j-1}+h^l_{i-1,j} \quad (i=1,\dots,n; j=1,\dots,m) \tag{4.21}$$

Constraint (4.20) imposes that, among all triangles, only one is used for the convex combination. Then, constraints (4.21) impose that the only α_{ij} values different from 0 can be those associated with the three vertices of such triangle.

4.2.3 Rectangle method

In this section we give a generalized description of a third method that was recently used by Borghetti, D'Ambrosio, Lodi and Martello [28] in the context of an application arising in electricity production. The idea is to improve the one-dimensional method through a correction term given by a better approximation on the y axis.

In this case too, let us introduce n coordinates $x_1, \dots, x_n$ on the x axis and m sampling coordinates $y_1, \dots, y_m$ on the y axis, with x_1 and x_n (resp. y_1 and y_m) coinciding with the left and right extremes of the x (resp. y) domain. For any given y value, say, $\overline{y} \in [y_j, y_{j+1})$, instead of associating a prefixed $\widetilde{y}_j$ to interval $[y_j, y_{j+1})$ (as in the one-dimensional method), we use the piecewise linear approximation of $f(x, y_j)$ with a linear correction depending on $\overline{y}$. More precisely, for a given point $(\overline{x}, \overline{y})$, with $\overline{x} \in [x_i, x_{i+1})$, the approximate value $f^a(\overline{x}, \overline{y})$ is given by:

$$f^a(\overline{x}, \overline{y}) = \lambda f(x_i, y_j) + (1-\lambda) f(x_{i+1}, y_j) + \delta \min\{\Delta(i,j), \Delta(i+1,j)\} \tag{4.22}$$

with $\Delta(l,j) = f(x_l, y_{(j+1)}) - f(x_l, y_j)$, and $\delta = (\overline{y} - y_j)/(y_{j+1} - y_j)$. The first two terms of the right-hand side are the extension of (4.2) and the third term is the correction. The meaning of the correction is better understood by relating the approach to the triangle method. Assume that the minimum in (4.22) occurs for $\Delta(i,j)$. By substituting we get:

$$f^a(\overline{x}, \overline{y}) = (\lambda - \delta) f(x_i, y_j) + (1-\lambda) f(x_{i+1}, y_j) + \delta f(x_i, y_{(j+1)}).$$

In other words, given the rectangle of vertices (x_i, y_j), (x_{i+1}, y_j), (x_{i+1}, y_{j+1}), (x_i, y_{j+1}) (see Figure 4.3), instead of approximating the function with the two induced triangles *abd* and *bcd*, as in Section 4.2.2, the function is evaluated using the rectangle *abcr* that lies on the plane passing through vertices $f(x_i, y_j)$, $f(x_{i+1}, y_j)$ and $f(x_i, y_{(j+1)})$ (b, c and a in Figure 4.3). If instead the minimum in (4.22) occurs for $\Delta(i+1, j)$, rectangle *sbcd* is used for the approximation.

We finally observe that equation (4.22) produces an underestimate with respect to the triangle approach, due to the "min" operator in the third term. If, according to the specific application, overestimate is desired, it is enough to substitute it with the "max" operator.

Let us now consider how the method can be modeled within a MILP. As in (4.9)-(4.13), let $\beta_1, \dots, \beta_{m-1}$ be a SOS1 with β_j taking the value 1 if $\overline{y}$ belongs to $[y_j, y_{j+1})$, and the value 0 otherwise. In addition let $\gamma_1, \dots, \gamma_{m-1}$ be continuous variables, taking values in the interval $[0,1]$. If $\overline{y} \in [y_j, y_{j+1})$ then $\gamma_j = (\overline{y} - y_j)/(y_{j+1} - y_j)$, and $\gamma_k = 0$ for all $k \neq j$. In other

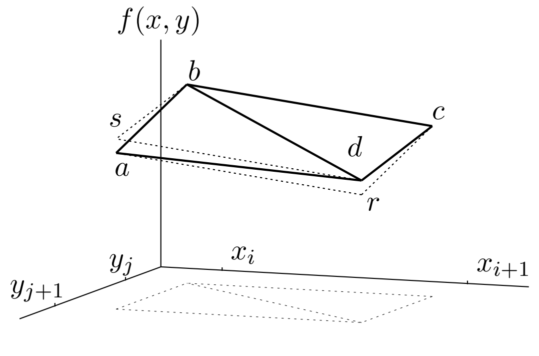


Figure 4.3: Geometric representation of the triangle method.

words, when y lies on the jth interval, γ_j represents the relative position of y within the jth interval. The approximate value f^a is then given by

$$(4.4), (4.5), (4.6), (4.7), (4.11)$$

$$y = \sum_{j=1}^{m-1} (\beta_j y_j + \gamma_j (y_{j+1} - y_j)) \tag{4.23}$$

$$\gamma_j \leq \beta_j \qquad (j = 1, \ldots, m-1) \tag{4.24}$$

$$f^a \leq \sum_{k=1}^{n} \alpha_k f(x_k, y_j) + \gamma_j K_{ij} + M(2 - \beta_j - h_i) \quad (j = 1, \ldots, m-1; \; i = 1, \ldots, n-1) \tag{4.25}$$

$$f^a \geq \sum_{k=1}^{n} \alpha_k f(x_k, y_j) + \gamma_j K_{ij} - M(2 - \beta_j - h_i) \quad (j = 1, \ldots, m-1; \; i = 1, \ldots, n-1), \tag{4.26}$$

where $K_{ij} = \min\{\Delta(i,j), \Delta(i+1,j)\}$ (see (4.22)). Due to the above definition of the β_j and γ_j variables, equations (4.23) and (4.24) impose that y is given by the unique non-zero term of the summation. Equations (4.25) and (4.26) are inactive when $\beta_j = 0$ or $h_i = 0$, hence providing $f^a = \sum_{k=1}^{n} \alpha_k f(x_k, y_j) + \gamma_j K_{ij}$ for the correct interval.

Note that, in order to keep the constraint matrix smaller in terms of non-zeros, one can efficiently reformulate constraints (4.25)–(4.26) by replacing the first term of the right-hand-side with a corresponding variable, say, φ_j, at the price of the addition of $m-1$ constraints. Namely, the final formulation is:

$$(4.4), (4.5), (4.6), (4.7), (4.11), (4.23), (4.24)$$

$$\varphi_j = \sum_{k=1}^{n} \alpha_k f(x_k, y_j) \qquad (j = 1, \ldots, m-1) \tag{4.27}$$

$$f^a \leq \varphi_j + \gamma_j K_{ij} + M(2 - \beta_j - h_i) \quad (j = 1, \ldots, m-1; i = 1, \ldots, n-1) \tag{4.28}$$

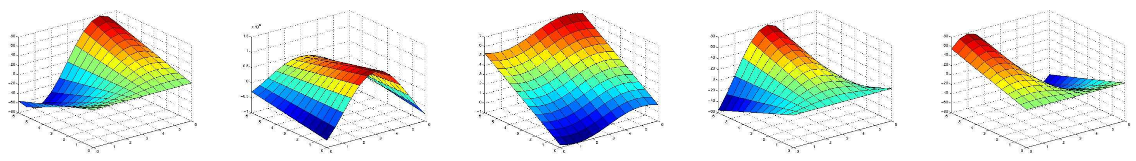
$$f^a \geq \varphi_j + \gamma_j K_{ij} - M(2 - \beta_j - h_i) \quad (j = 1, \ldots, m-1; i = 1, \ldots, n-1). \tag{4.29}$$

4.3 Comparison

In this section we discuss computational issues associated with the three approaches for the piecewise approximation of a function of two variables. In particular, in Section 4.3.1 we show that none of the three techniques dominates any other in terms of quality of the given approximation. In Section 4.3.2 we discuss the embedding of the three techniques within a MILP model, thus taking into account other indicators besides the quality of the approximation, such as for example the size of the corresponding models.

4.3.1 Dominance and approximation quality

We first show that the three methods do not dominate each other. Consider function $f_1(x, y) = y(\sin(\pi(x-3)/4)))$ (Figure 4.4(a)), and its approximation in the range $x \in [1, 5]$, $y \in [1, 5]$ with $n = m = 5$. For $(\overline{x}, \overline{y}) = (3.5, 1.5)$, the actual value is 0.57: the approximate value computed by both the one-dimensional and the rectangle method is 0.35, while



(a) $f_1(x, y)$ (b) $f_2(x, y)$ (c) $f_3(x, y)$ (d) $f_4(x, y)$ (e) $f_5(x, y)$

Figure 4.4: Five functions used to evaluate the approximation quality.

the triangle method provides the best approximation 0.71. For $(\overline{x}, \overline{y}) = (1.51, 4.65)$, the actual value is -4.29: the best approximation, -4.06, is provided by the rectangle method, while the one-dimensional and the triangle method give -3.40 and -3.91, respectively. Consider now function $f_2 = (10 - y)^3(\sin(\pi(x - 1)/4)))$ (Figure 4.4(b)) in the same range. For $(\overline{x}, \overline{y}) = (1.5, 1.5)$, the actual value is 235.02: the approximate value computed by the one-dimensional method is 257.74, which is better than the value 181.02 produced by the other two methods.

Let us now consider the average quality of the approximation provided by the three methods. We used the two functions above and three additional functions (all shown in Figure 4.4). The results are reported in Table 4.1. Each entry gives an average value computed over 40 tests obtained by varying the values of n, m, $\overline{x}$ and $\overline{y}$. (Functions definitions and experiment settings are available on demand from the authors.) For each function and for each method the table gives the average percentage approximation error (computed as $100 \cdot |f^a(\overline{x}, \overline{y}) - f(\overline{x}, \overline{y})|/f(\overline{x}, \overline{y})$) and the percentage of times the method gave the best approximation.

The triangle method provides on average a tighter approximation with respect to the other techniques (without, however, dominating them, see, e.g., function $f_1(x, y)$), as it could be expected in view of its higher complexity. Such a complexity also implies drawbacks concerning its tractability when embedded in MILP models, as shown in the next section on the basis of computational experiments on a real-world application.

Table 4.1: Average approximation quality for different values of n, m, $\overline{x}$ and $\overline{y}$.

	% error			% best		
function	one-dimensional	triangle	rectangle	one-dimensional	triangle	rectangle
$f_1(x, y)$	20.38	9.27	9.92	4.88	53.66	56.10
$f_2(x, y)$	20.15	9.27	27.38	29.27	90.24	39.02
$f_3(x, y)$	18.44	3.87	3.87	9.76	100.00	100.00
$f_4(x, y)$	20.45	8.76	15.09	19.51	100.00	31.71
$f_5(x, y)$	20.38	9.27	9.77	24.39	87.80	82.93

4.3.2 Computational experiments

In order to experimentally evaluate the three approaches on instances of realistic size, we considered the following real-world problem. The *Short-Term Hydro Scheduling* is the problem of finding the optimal scheduling of a multi-unit hydro power station. In a short-term time horizon one wants to maximize the revenue given by power selling. We suppose that the generation company acts as a price-taker, and that the electricity prices and the inflows are forecasted. The non-linear part of this problem is the power function which depends on the water flow and the basin volume. We approximated this function of two variables using the methods presented in the previous sections. (For a more detailed discussion on this application the reader is referred to [28].)

We considered a simplified version of a specific instance of the problem discussed in [28] with 168 time periods to be planned. The simplification consists in not considering the possibility to pump water in the basin and has the goal to allow a fair comparison of the three methods, purged by the influence of big-M type constraints that is needed if the pump-storage is considered.

Table 4.2 reports, for each model and for each pair (n, m), the size of the corresponding MILP models, namely, the number of variables (overall and binaries), constraints and non-zeros.

It is easy to see that one-dimensional and rectangle methods give rise to models which are comparable in terms of number of variables and constraints, although the latter has slightly more of both. The rectangle method has the smallest number of non-zero entries, much smaller than that of the one-dimensional method and about half that of the triangle method which, in addition, has by far the largest number of variables (both continuous and binary). However, these numbers do not necessarily give a full picture of the impact of the three methods on the solution of the associated MILPs.

Table 4.3 reports the results obtained by running ILOG-Cplex 10.2 on the resulting MILPs. The table gives, for each pair (n, m) and for different time limits depending on the size of the corresponding problems: (i) the solution value obtained at the time limit (or earlier, if the MILP solver could prove optimality), recomputed using the original non-linear function; (ii) the percentage error of such a value with respect to the one computed by the solver; (iii) the initial and final percentage gaps; (iv) the CPU time in seconds (or "T.L." when the time limit occurred); (v) the number of Branch-and-Bound nodes. The initial percentage gap was computed using the initial value of the linear programming relaxation, say, UB_i, and the final value of the best feasible solution, say, LB. The final percentage gap was computed using the final upper bound, say, UB_f, and again the best feasible solution value. Both gaps were computed as $100 \cdot (UB - LB)/LB$. When no feasible solution was found by the MILP solver within the time limit, the table reports "n/a" ("not available").

The results show a number of interesting facts:

Despite the huge number of non-zero entries, the one-dimensional method is very fast in closing the initial (rather large) gap both in terms of CPU time and number of nodes. This is partially due to the fact that the preprocessing phase of ILOG-Cplex 10.2 is very effective in significantly reducing the size of the model.

On the contrary, the MILPs associated with the triangle method are hard to solve although the initial percentage gap is fairly small. (Only one instance solved to optimality in the time limit.)

The difficulty with the triangle method is really the size of the corresponding MILPs. Indeed, in the cases in which both the triangle and the rectangle method are unable to close the gap within the time limit, one can observe that the latter explores many more nodes than the former (up to one order of magnitude more for the $(50, 50)$ case).

The rectangle method is unable to fully close the initial (fairly large) percentage gap in only one instance if the 1 hour time limit is allowed.

The quality of the solutions returned by the one-dimensional method is generally not good. The percentage error of the computed solution value with respect to the real value is significantly higher than that produced by the other methods.

The rectangle method takes advantage of the increase of both n and m by returning strictly better solutions. This is not always the case for the one-dimensional method for which the same optimal solution is returned for both cases $(30, 30)$ and $(40, 40)$.

As somehow expected by the analysis of Section 4.3.1, the percentage error of the triangle method is generally the smallest one.

The numbers and the analysis are confirmed by different instances with slight variations. There are of course instances in which the triangle method not only gives a better approximation, but is also fast enough. This is probably the case when the optimal solution is very tightly approximated by the triangles and a small set of alternative quasi-optima is present. However, the trend previously discussed does not change significantly.

Overall, the three methods all present advantages and drawbacks and a careful analysis of the specific application should be used to decide which method is particularly suited for it. This also strategically depends on the tradeoff between the quality of the approximation (although as discussed no dominance can be proven in general) and the computational effort one is ready to spend. It seems that the rectangle method offers a rather good compromise for such a tradeoff and a quite stable computational behavior.

Acknowledgments

This work was supported by Università di Bologna, Project DecisOpElet 2006.

Table 4.2: Comparison with respect to the size of the MILP.

		one-dimensional method				triangle method				rectangle method			
		# variables				# variables				# variables			
n	m	all	binary	# constraints	# non-zeros	all	binary	# constraints	# non-zeros	all	binary	# constraints	# non-zeros
10	10	6,161	3,810	22,008	219,295	51,575	34,104	18,816	229,483	9,467	3,810	25,482	121,365
20	20	11,138	7,107	75,768	1,509,316	202,775	134,904	69,216	925,003	17,741	7,107	82,539	438,384
30	30	16,115	10,404	163,128	4,881,277	454,775	302,904	153,216	2,090,923	26,015	10,404	173,196	954,483
40	40	21,101	13,710	284,088	11,343,565	807,575	538,104	270,816	3,727,243	34,307	13,710	297,462	1,670,445
50	50	26,078	17,007	438,648	21,903,496	1,261,175	840,504	422,016	5,833,963	42,581	17,007	455,319	2,584,884

Table 4.3: MILP results with different time limits expressed in CPU seconds.

			one-dimensional method						triangle method						rectangle method					
		Time	solution	%	initial	final	CPU	#	solution	%	initial	final	CPU	#	solution	%	initial	final	CPU	#
n	m	Limit	value	error	%gap	%gap	time	nodes	value	var	%gap	%gap	time	nodes	value	var	%gap	%gap	time	nodes
10	10	300	31,576.30	-3.80	12.25	—	2.00	158	31,576.30	-2.55	1.49	0.22	T.L.	7,684	31,576.30	-2.30	11.25	—	21.61	30,850
		600	31,576.30	-3.80	12.25	—	2.00	158	31,576.30	-2.55	1.49	—	304.69	13,542	31,576.30	-2.30	11.25	—	21.61	30,850
20	20	300	31,611.60	-2.78	8.82	—	4.89	108	n/a	n/a	n/a	n/a	T.L.	1,121	31,613.80	-2.36	8.80	0.02	T.L.	13,557
		600	31,611.60	-2.78	8.82	—	4.89	108	31,555.10	-2.33	1.40	0.60	T.L.	3,699	31,613.80	-2.36	8.80	0.01	T.L.	29,978
		3,600	31,611.60	-2.78	8.82	—	4.89	108	31,582.00	-2.30	1.29	0.41	T.L.	35,382	31,613.80	-2.36	8.80	—	801.99	33,833
30	30	300	31,629.30	-2.69	10.62	—	12.59	441	n/a	n/a	n/a	n/a	T.L.	411	31,629.20	-2.34	10.53	0.05	T.L.	1,330
		600	31,629.30	-2.69	10.62	—	12.59	441	n/a	n/a	n/a	n/a	T.L.	1,285	31,630.50	-2.34	10.52	0.01	T.L.	5,472
		3,600	31,629.30	-2.69	10.62	—	12.59	441	31,475.10	-2.34	1.79	0.84	T.L.	4,310	31,630.50	-2.34	10.52	—	780.84	6,017
40	40	3,600	31,629.30	-2.61	13.10	—	37.56	1,080	n/a	n/a	n/a	n/a	T.L.	3,787	31,636.60	-2.33	12.88	—	1,534.82	6,370
50	50	3,600	31,636.80	-2.51	11.57	—	218.12	10,533	n/a	n/a	n/a	n/a	T.L.	1,697	31,639.50	-2.34	11.47	0.50	T.L.	11,354

Chapter 5

NLP-Based Heuristics for MILP problems

[1]

This chapter is dedicated to the description of ideas and preliminary computational results on heuristics for Mixed Integer Linear Programming. The relationship between this topic and this Ph.D. thesis will be clear soon.

Let us consider a standard MILP problem (see Section 1.1):

$$\begin{aligned} \min c^T x + d^T y & & \\ Ax + By &\leq& b \\ x &\in& X \cap \mathbb{Z}^n \\ y &\in& Y. \end{aligned}$$

The feasible region of the continuous relaxation of this MILP is a convex set, more precisely a polyhedron. However, the MILP problem presents a source of discontinuity and non-convexity, represented by the integrality requirements. The feasibility problem, i.e. the problem aimed at finding a feasible solution for MILP, is NP-hard (see [61]) and, sometimes, also a hard task in practice. The feasibility problem can be mapped into the following Non-Linear Programming (NLP_f) problem:

$$\min f(x) \tag{5.1}$$

$$Ax + By \leq b \tag{5.2}$$

$$x \in X \tag{5.3}$$

$$y \in Y, \tag{5.4}$$

with $f : \mathbb{R}^n \to \mathbb{R}$ being a function such that:

$$f(x) = \begin{cases} 0 & \text{if } x \in \mathbb{Z}^n \\ > 0 & \text{otherwise.} \end{cases}$$

[1]This chapter summarizes ideas and discussions arising for an ongoing project involving Matteo Fischetti (DEI, University of Padova), Antonio Frangioni (DI, University of Pisa), Andrea Lodi (DEIS, University of Bologna) and Andreas Wächter (Department of Mathematical Sciences, IBM T.J. Watson Research Center, Yorktown Heights, NY).

In the case $x \in \{0,1\}^n$, i.e. all the integer variables are binary, an example of $f(x)$ can be:

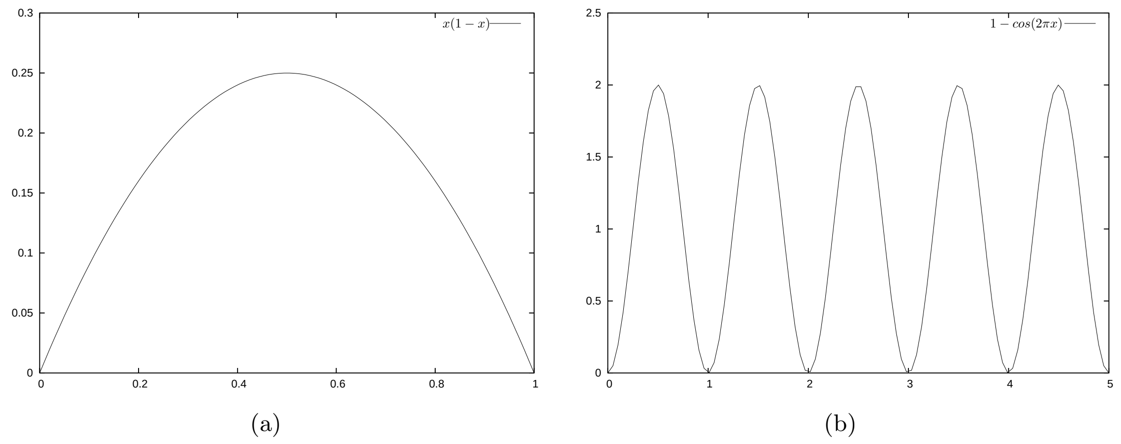
$$f(x) = \sum_{j=1}^{n} (x_j(1 - x_j)).$$

The plot of a single term of the summation is depicted in Figure 5.1 (a). It is a non-linear concave function. In the general integer case, i.e. some of the integer variables are not binary, it is possible to use, for example, the non-linear and non-convex function of Figure 5.1 (b):

$$f(x) = \sum_{j=1}^{n} (1 - \cos(2\pi x_j)).$$

The resulting problem NLP_f is a non-convex NLP. In the context of this Ph.D. thesis, we

Figure 5.1: Examples of $f(x)$ for (a) binary and (b) general integer variables.



tried to apply Non-Linear Programming techniques to MILP feasibility problem. It is well-known that solving NLPs is a complicated issue, but the large improvements which involved NLP solvers in the latest years might be exploited, together with MILP techniques, to solve the MILP feasibility problem. However, as it is clear from Figure 5.1, this non-convex NLP problem could be very nasty to deal with, expecially because of the presence of local minima. For the moment we consider all the global minima of the function $f(x)$ good, i.e. the minima for which $f(x)$ is 0, because we are looking for any feasible solution. The chapter is organized as follows: in Section 5.1 we describe the Frank-Wolfe method [57], aimed at finding the solution of an NLP problem with linear constraints and a quadratic objective function. We show the relationship between this algorithm and the Feasibility Pump method [50], aimed at finding heuristic solutions for MILP problems. In Section 5.2 we present computational results on using NLP solvers to solve the problem NLP_f and in Section 5.3 we show the importance of diversification and randomness techniques. We apply standard MILP methods in this context, such as cuts, to improve results (Section 5.4). Finally, in Section 5.5, we draw conclusions and discuss future work directions.

5.1 The NLP problem and the Frank-Wolfe Method

The problem NLP_f has a special structure. In particular, all the constraints are linear, leading to a feasible region equivalent to a polyhedron. The only non-linearity present in the model is the objective function. Special-purpose algorithms for this kind of NLP problems were studied for more than fifty years. An example of such algorithms is the Frank-Wolfe (FW) method, introduced in 1956 [57], which was originally proposed for problems with a non-linear quadratic objective function (as in the binary special case above).

Let us consider the problem NLP_f in the simplified form:

$$\begin{gathered}\min f(x,y)\\ (x,y)\in S,\end{gathered}$$

where S is a polyhedron (in our case, $S = \{(x,y) \mid Ax + By \le b,\ x \in X,\ y \in Y\}$ and $f(x,y) = f(x)$). The Frank-Wolfe algorithm solves this kind of problems (see Algorithm 4). At step 2, an MILP problem is solved: z represents a vector of $n \times m$ variables and z^k is the

Algorithm 4 The Frank-Wolfe algorithm.

```
1: Initial solution, $(x^0, y^0) \in S$. Set $k := 0$.
2: At each iteration $k$, determine a search direction, say, $p^k$, solving an approximation of the
   original problem obtained by replacing the function $f$ with its first-order Taylor expansion
   around $(x^k, y^k)$:
      $\min \; f(x^k,y^k) + \nabla f(x^k,y^k)^T (z - \begin{bmatrix} x^k \\ y^k \end{bmatrix})$
      $z \in S$,
   set $p^k := z^k - \begin{bmatrix} x^k \\ y^k \end{bmatrix}$.
3: Determine a step length $\alpha^k$, solving
      $\min_{\alpha\in[0,1]} f(\begin{bmatrix} x^k \\ y^k \end{bmatrix} + \alpha p^k)$.
4: New iteration point:
      $(x^{k+1}, y^{k+1}) = \begin{bmatrix} x^k \\ y^k \end{bmatrix} + \alpha^k p^k$.
5: if terminating_condition then
6:    Stop!
7:    ($(x^{k+1}, y^{k+1})$ is an approximation of $(x^{opt}, y^{opt})$.)
8: else
9:    Set $k := k + 1$ and iterate 2.
10: end if
```

solution of the MILP problem at iteration k. At step 3, an unconstrained NLP problem is solved: α is the unique variable, which is only bounded in the range $[0, 1]$.

Let us consider the binary case, say, NLP_{f01}, for which we use $f(x) = \sum_{j=1}^{n}(x_j(1 - x_j))$. The Frank-Wolfe algorithm seems to be suitable for NLP_{f01} and a modified version of it is given by Algorithm 5. Note that, even if y^k is not present in the objective function of the

Algorithm 5 The Frank-Wolfe algorithm for NLP_{f01}.

```
1: Initial solution, $(x^0, y^0) \in S$. Set $k := 0$.
2: At each iteration $k$, determine a search direction, say, $p^k$, solving an approximation of the
   original problem obtained by replacing the function $f$ with its first-order Taylor expansion
   around $(x^k, y^k)$:
       $\min \sum_{j=1}^{n}(x_j^k(1 - x_j^k)) + \sum_{j=1}^{n}(1 - 2x_j^k)(z_j - x_j^k)$
       $z \in S,$
   set $p^k := z^k - \begin{bmatrix} x^k \\ y^k \end{bmatrix}$.
3: Determine a step length $\alpha^k$, solving
       $\min_{\alpha \in [0,1]} \sum_{j=1}^{n}(x_j^k + \alpha p_j^k)(1 - (x_j^k + \alpha p_j^k)).$
4: New iteration point:
       $(x^{k+1}, y^{k+1}) = \begin{bmatrix} x^k \\ y^k \end{bmatrix} + \alpha^k p^k.$
5: if terminating_condition then
6:    Stop!
7:    ($(x^{k+1}, y^{k+1})$ is an approximation of $(x^{opt}, y^{opt})$.)
8: else
9:    Set $k := k + 1$ and iterate 2.
10: end if
```

problem of step 2, the direction is taken also for the m components of z which are involved in the constraints. Observe that the problem of step 2, in the case that x^k is binary, can be rewritten in the following way:

$$\begin{aligned} \min \quad & \sum_{j \in \{1,\dots,n\}: x_j^k = 0} z_j + \sum_{j \in \{1,\dots,n\}: x_j^k = 1} (1 - z_j) \\ & z \in S. \end{aligned}$$

The objective function resembles the one used for the Feasibility Pump (FP) algorithm, see [50]. This algorithm has similarities with the FW algorithm. In Algorithm 6 we present the FP algorithm in order to compare it with the particular FW method given in Algorithm 5 (see also Eckstein and Nediak [46] for further considerations about the relationship between these two algorithms). Note that the terminating condition include both iterations/time limit and the feasibility test, i.e. if the solution is feasible for the MILP problem, the algorithm stops.

The main difference between the two methods is that the FW algorithm generates a sequence of points (x^k, y^k) which stays inside the set S and the FP algorithm generates two

Algorithm 6 The Feasibility Pump (heuristic) algorithm for MILP problems with binary variables.

```
1: Initial solution, $(x^0, y^0) \in S$. Set $k := 0$.
2: Solve
       $$\min \sum_{j \in \{1,\dots,n\}: x_j^k = 0} z_j + \sum_{j \in \{1,\dots,n\}: x_j^k = 1} (1 - z_j)$$
       $$z \in S$$
   obtaining $z^k$.
3: New iteration point:
       $$x_j^{k+1} = [z_j^k] \ \forall j \in \{1,\dots,n\},$$
   and
       $$y_j^{k+1} = z_{j+n}^k \ \forall j \in \{1,\dots,m\},$$
   where $[z_j^k]$ is the rounding to the nearest integer of the $j$-th component of solution $z^k$ of
   the previous step.
4: if $x^{k+1} = x^k$ then
5:     Flip the rand$(T/2, 3T/2)$ entries $x_j$ $(j = 1,\dots,n)$ with highest $|z_j^k - x_j^{k+1}|$.
6: end if
7: if terminating_condition then
8:     Stop!
9:     (If no iterations/time limit was reached, $(x^{k+1}, y^{k+1})$ is a feasible solution for MILP.)
10: else
11:     Set $k := k + 1$ and iterate 2.
12: end if
```

sequences of points z^k and (x^k, y^k). The first sequence stays inside the set S and can be viewed as the sequence generated by FW always taking the maximum step length possible, i.e. $\alpha = 1$. The second sequence always stays outside the set S until a MILP feasible solution is found, but always satisfies the integrality requirement. We can imagine this second sequence as generated by a FW method applied to a relaxation of the MILP problem:

$$\min \sum_{j \in \{1,\dots,n\}: x_j^k = 0} z_j + \sum_{j \in \{1,\dots,n\}: x_j^k = 1} (1 - z_j)$$
$$z \in \{0,1\}^{n+p}.$$

This is a sort of diversification step with respect to the FW method. Also step 5 is a diversification phase: when the algorithm cycles ($x^{k+1} = x^k$), the solution is modified randomly. It is a random restart to go far away from a cycling situation.

Now we want to understand if these considerations can be extended to the general integer case. Let us consider the function:

$$f(x) = \sum_{j=1}^{n} (1 - \cos(2\pi x_j)).$$

Table 5.1: Comparison among different NLP solvers used for solving problem NLP_f.

Solver	# solutions found	# not terminated	# non integer
filter	13	13	34
ipopt	13	7	40
knitro	14	6	40
lancelot	6	28	26
loqo	6	34	20
minos	12	13	35
pennon	15	13	32
snopt	12	13	35

The objective function of step 2 of Algorithm 4 would be:

$$\min \sum_{j=1}^{n} (1 - \cos(2\pi x_j^k)) + \sum_{j=1}^{n} (2\pi \sin(2\pi x_j^k)(z_j - x_j^k)) = 0$$

for x^k integer, so the straightforward extension to the general integer case seems not to be interesting.

The FP algorithm was extended to the general integer case by Bertacco et al. [16]. In that case the relationship between FP and FW is by far less clear and the FP algorithm needs more involved (and sometimes time consuming) mechanism than in the binary case.

5.2 Solving NLP_f directly by using different NLP solvers

The first tests were performed using NLP solvers to solve the NLP_f problem. Different types of NLP solvers are available (see Chapters 1 and 8 for details) and can be used. We decided to try to compare the performance of the different NLP solvers implementing the model under AMPL environment (see Chapter 8) and using the 60 instances of the standard Miplib2003 library (see [4]) as testbed. We used AMPL in order to have the same interface for different NLP solvers which can be accessed through the Kestrel feature of NEOS, the server for Optimization from which the user can access to different kind of solvers for different classes of optimization problems (see [96] and Chapter 8 for details).

In Table 5.1 we reported the results for 8 different NLP solvers using their default options. We reported the name of the solver (Solver), the number of instances for which a MILP feasible solution was found, over 60 instances (# solutions found), the number of instances for which the solver encountered numerical or memory difficulties (# not terminated) and the number of instances for which no MILP feasible solution was found within the time limit (# non integer). From the results it is clear that the solvers which performs better are knitro [33] and ipopt [123] because they provide a good compromise between number of MILP feasible solution found and reliability from a memory/numerical viewpoint. On the other hand, also pennon is able to find a high member of feasibility solutions but has a less stable behavior, i.e. almost double number of instances in which encountered numerical or memory difficulties. In the next section we will use the trunk version of ipopt to perform our tests because we prefer to use an open-source software in order to be able to control and modify it for our special class of problems.

5.3 The importance of randomness/diversification

In Feasibility Pump the randomness plays a fundamental role: step 5 of Algorithm 6 is performed for the 66% of the instances. In order to fairly compare our results with the FP ones, we decided to add the randomness/diversification ingredient in our method using different starting points. They, in general, depends on the value of the continuous (LP) relaxation of the MILP problem:

sp_1: The optimal solution of the continuous relaxation of the MILP obtained using the baropt algorithm.

sp_2: Like sp_1, but with other options.

sp_3: Like sp_1, but using the dualopt algorithm.

sp_4: Like sp_1, but using the lpopt algorithm.

sp_5: Like sp_1, but using the primopt algorithm.

sp_6: The combination of the optimal solution of the continuous relaxation of the MILP and the optimal solution of the continuous relaxation of the MILP in which the objective function has been inverted, using a 0.5 weight for the solutions (see Figure 5.2). The algorithm used is baropt.

sp_7: Like sp_6, but with other options.

sp_8: Like sp_6, but using a randomly defined weight.

sp_9: Like sp_8, but using a randomly defined weight for each element of the solution.

sp_10: A randomly generated solution.

Figure 5.2: sp_6-sp_9 are the combination of solutions (1.4, 1.2) and (3.2, 3.7) represented by one point of the line linking the two points.

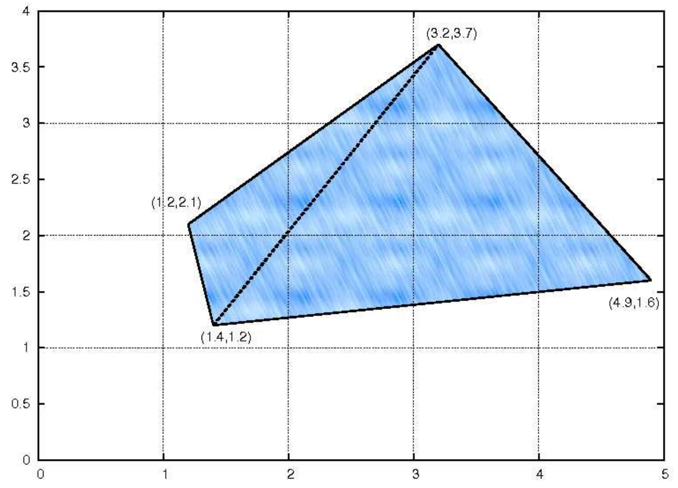


Figure 5.2 depicts the feasible region of a simple problem, i.e. the light blue area. Assume that points $(1.4, 1.2)$ and $(3.2, 3.7)$ are the optimal solutions of the continuous relaxation

of the MILP problem and of the continuous relaxation in which the objective function has been inverted, respectively. The point of the black segment linking the two points are the possible starting points, obtained using different weights for the combination. In Table 5.2 we report, in the first part of the table, the results at iteration 1, i.e. just solving one NLP_f problem, using one of the starting points presented. The second part of the table (the last five rows) shows the results for sp_10 at different iterations, from 1 to 8. At each iteration a different (randomly generated) starting point is given to the NLP solver. The first column is the starting point type used, the second the number of iterations which the results are referred to, then the number of instances for which a MILP feasible solution was found (# solutions found), for which the NLP solver encountered numerical/memory problems (# not terminated) and for which no MILP feasible solution was found within the time limit (# non integer). The results show that changing the starting point can make a relevant difference.

Table 5.2: Results using different starting points.

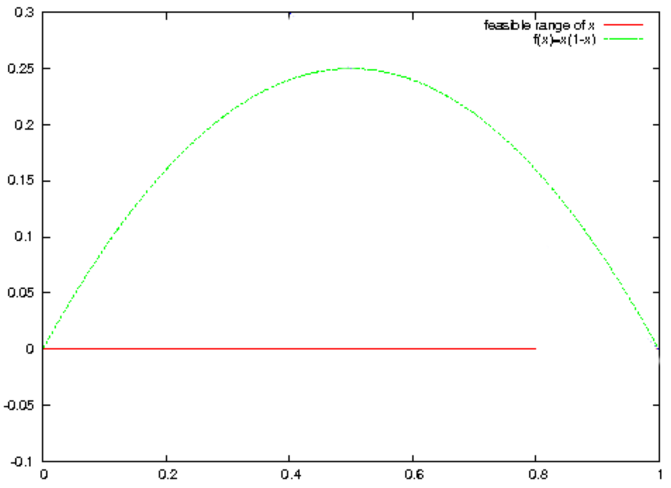
Starting Point	iter #	# solutions found	# not terminated	# non integer
sp_1	1	5	3	52
sp_2	1	9	3	48
sp_3	1	8	3	49
sp_4	1	8	4	48
sp_5	1	6	3	51
sp_6	1	23	4	33
sp_7	1	21	11	28
sp_8	1	21	4	45
sp_9	1	17	5	38
sp_10	1	14	-	-
	3	16	-	-
	5	18	-	-
	6	19	-	-
	8	21	3	37

In particular, very good results are obtained using the combination of the optimal solution of the LP relaxation and of the "inverted-direction" objective function LP relaxation. This can be explained in the following way: the combination of these two solutions has a high chance to be in the interior of the polyhedron. This means that the local solution obtained by the NLP solver has a higher chance to be a MILP feasible point or close to it, because the starting point is not necessary in the boundary like in cases sp_1-sp_5. Of course, the solution found in this way might be not very close to the optimal solution of the MILP problem, but this is not what we are concerned at the moment.

5.4 Apply some MILP techniques

If no problems are encountered during the NLP execution, the provided solution satisfies all the constraints, but the integer requirements. In the previous section we proposed to give to the NLP solver, at each iteration, a different starting point in order to end up with a different solution. Another possibility, taken from MILP methods, to obtain a different solution in the next iteration is adding a valid cut. In particular, we generate a Chvátal-Gomory cut (see [52]) cutting off the fractional solution obtained at iteration k by the solver. Cutting that solution would lead us, in the next iteration, to a solution which is different from the previous one. Figure 5.3 shows the plot of $x_j(1 - x_j)$ for a single term x_j supposed to be binary. We suppose also that, considering the constraints of our MILP problem, the feasible range of x_j is $[0, 0.8]$. If the starting point given to the NLP solver is, for example, 0.6, the NLP solver might end up with a solution with $x_j = 0.8$. At this point we would like to find a valid cut which excludes for x_j the range $[0.5, 0.8]$ so as the NLP solver would naturally go to the local minimum to $x_j = 0$ in the next iteration. Of course, this would be the ideal case and the Chvátal-Gomory cut is in general not doing the job in just one shot.

Figure 5.3: An ideal cut should make the range $[0.5, 0.8]$ infeasible.



In Tables 5.3 and 5.4 we show the behavior of our experiments for instances `gesa2-o` and `vpm2`, respectively: at each iteration a Chvátal-Gomory cut is added when a fractional solution is provided by the NLP solver. In the first table the impact obtained by adding a cut on the value of $f(x)$ is evident. Function $f(x)$ gives a measure of the integer infeasibility of the solution provided by the NLP solver and, as shown in Table 5.3, at each iteration it decreases and at iteration 3 a MILP feasible solution is found. Table 5.4 shows that adding a cut can also bring to an increase of the infeasibility of the solution provided by the NLP, but also in this case a MILP solution is found in few iterations. Of course, this technique is not always as effective as in these two cases, however we think that combining NLP and MILP techniques can lead to a good exploitation of the advantages of both the methods.

Table 5.3: Instance `gesa2-o`

Iteration #	$f(x)$
1	0.818001
2	0.091539
3	0.000001

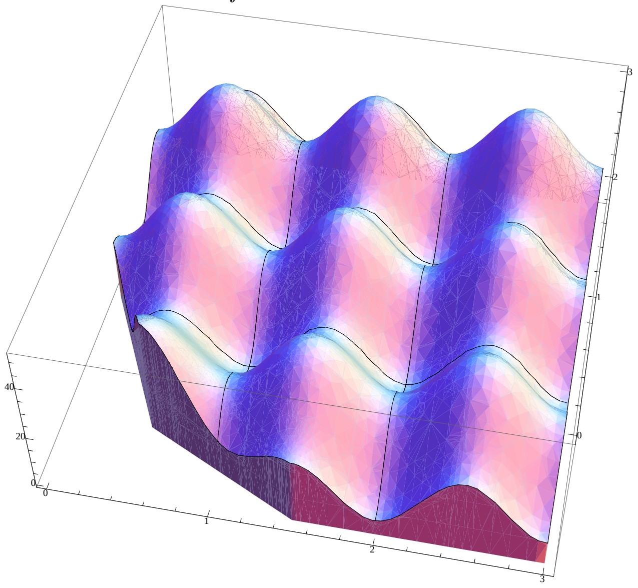
Table 5.4: Instance `vpm2`

Iteration #	$f(x)$
1	0.064632
2	0.166161
3	0.145310
4	0.000000

5.5 Final considerations and future work

In the previous sections we experimented on how to integrate NLP and MILP techniques to solve the MILP feasibility problem. We proposed to treat the intrinsic non-convexity of MILP problems as a non-linear objective function, obtaining a non-convex NLP problem, and observed similarities among the Feasibility Pump algorithm, addressed to MILP problems, and the Frank-Wolfe algorithm, addressed to NLP problems with linear constraints. We experimented different solvers to obtain a solution of the non-convex NLP. Using an NLP solver with different starting points (see Section 5.3) we obtained a MILP feasible solution for 29 over the 60 instances of the Miplib2003 (5 over 15 considering only the general integer instances). For 24 over the 45 binary instances we were able to find a MILP feasible solution at the first iteration. This happens for Feasibility Pump heuristic only 8 times. Of course, an iteration of our algorithm is much more expensive than an FP iteration, but improvements on this side are possible. Another potential issue is that our problem NLP_f can have lots of local minima (see, for example, Figure 5.4) which are not global optima, i.e. they do not have value 0. When the NLP solver ends up with a fractional solution, one possibility is to rerun the NLP solver with a different starting point. Another possibility to escape from local minima is applying MILP techniques to cut them in the next iterations, as shown in Section 5.4.

Figure 5.4: NLP_f can have lots of local minima.



We presented preliminary results, with the main goal of understanding the relationship among the MILP and NLP building blocks and their intrinsic difficulties. First of all we want to focus on the general integer instances, because, for the binary case, very effective heuristics have already been proposed. We plan to try to identify specialized cuts, linear or non-linear because we solve an NLP problem without exploiting, for the moment, the fact that the constraints of the NLP are all linear. As seen from the computational results, the randomness/diversification step plays a fundamental role. For this reason, it would be interesting to study different and more specific techniques to systematically exploit randomness and diversification. The results presented in Section 5.3 using different starting points were useful also to observe that using starting points which stay in the interior of the feasible region helps the solver. Considering this, it would be useful helping the solver to go away from the barrier of the feasible region penalizing in some way the points lying in it. In a second phase, as we already mentioned, we want to diversify the different global optima, i.e. do not treat in the same way all the MILP feasible solutions, but penalize the solutions which have a large value of the original objective function.

Part III

Applications

Chapter 6

Hydro Scheduling and Unit Commitment

[1]

Nomenclature

Sets

$T = \{1, \dots, \overline{t}\}$ = set of time periods considered;
$J = \{1, \dots, \overline{n}\}$ = set of turbine-pump units.

Parameters

I_t = predicted water inflow in period t $(t \in T)$ [m^3/s];
Π_t = unit price of the power generated/consumed in period t $(t \in T)$ [€/MWh];
Δt = period duration [hours];
C_j = startup cost of unit j as a turbine $(j \in J)$ [€];
D_j = startup cost of unit j as a pump $(j \in J)$ [€];
$\underline{Q}_j, \overline{Q}_j$ = min and max flow value in turbine j $(j \in J)$ (when the turbine is on) [m^3/s];
$\overline{P}_j$ = max power produced by turbine j $(j \in J)$ [MW];
$\Delta q^-, \Delta q^+$ = max ramp down and ramp up [m^3/s];
$\underline{V}, \overline{V}$ = min and max water volume in the basin [m^3];
V_0 = water volume in the basin in period 0 [m^3];
$V_{\overline{t}}$ = target (final) water volume in the basin [m^3];
$\overline{S}$ = max water spillage [m^3/s];
W_j = water needed to start up turbine j $(j \in J)$ [m^3/s];
Y_j = water needed to start up pump j $(j \in J)$ [m^3/s];
E_j = energy needed to start up pump j $(j \in J)$ [MWh];
Q_{j0} = flow in turbine j in period 0 $(j \in J)$ [m^3/s];
G_{j0} = status of turbine j in period 0 $(j \in J)$ [1 on, 0 off];

[1]The results of this chapter appears in: A. Borghetti, C. D'Ambrosio, A. Lodi, S. Martello, An MILP Approach for Short-Term Hydro Scheduling and Unit Commitment with Head-Dependent Reservoir, *IEEE Transactions on Power Systems*, 23(3), 1115–1124, 2008 [28].

U_{j0} = status of pump j in period 0 ($j \in J$) [1 on, 0 off];
Q_j^- = flow pumped by pump j ($j \in J; Q_j^- < 0$) [m^3/s];
P_j^- = power consumed during pumping by pump j ($j \in J; P_j^- < 0$) [MW];
$\underline{\Theta}$ = min released water in each period [m^3/s].

Variables

q_{jt} = water flow in unit j in period t ($j \in J, t \in T$), with $q_{j0} = Q_{j0}$ [m^3/s];
v_t = water volume in the basin in period t ($t \in T$), with $v_0 = V_0$ [m^3];
p_{jt} = power generated or consumed by unit j in period t ($j \in J, t \in T$) [MW];
s_t = spillage in period t ($t \in T$) [m^3/s];
w_{jt} = shutdown phase of turbine j in period t ($j \in J, t \in T$) [1 if it is shutdown, 0 otherwise];
$\widetilde{w}_{jt}$ = startup phase of turbine j in period t ($j \in J, t \in T$) [1 if it is started up, 0 otherwise];
g_{jt} = status of turbine j in period t ($j \in J, t \in T$), with $g_{j0} = G_{j0}$ [1 on, 0 off];
y_{jt} = shutdown phase of pump j in period t ($j \in J, t \in T$) [1 if it is shutdown, 0 otherwise];
$\widetilde{y}_{jt}$ = startup phase of pump j in period t ($j \in J, t \in T$) [1 if it is started up, 0 otherwise];
u_{jt} = status of pump j in period t ($j \in J, t \in T$), with $u_{j0} = U_{j0}$ [1 on, 0 off].

Some additional parameters and variables, introduced to linearize the model, are defined in Section 7.2.

6.1 Introduction

We consider a price-taker generating company that wants to optimize the operation of a pump-storage multi-unit hydro power station for a given time horizon, typically one day or one week. The problem is to determine the commitment and the power generation of the plant so as to maximize the revenue given by power selling. All the units of the plant are assumed to be fed by the same reservoir. We assume that inflows and prices are known as previously forecasted.

Several approaches have been proposed for the solution of this problem. For an exhaustive overview we refer the reader to the recent survey [103]. In [106] the problem was formulated as a simple Linear Programming (LP) model by neglecting costs and constraints relevant to start-ups and shutdowns. In [35] a Non-Linear Programming (NLP) model with some simplified assumption was introduced. Ad-hoc heuristics were proposed by several authors, such as [117] and [101]. In [88] a multistage looping optimization algorithm was proposed for the development of the optimal bidding strategies of an individual pumped-storage unit owner in a competitive electricity market. In [49] the large-scale mixed-integer NLP problem of determining the optimal scheduling of hydropower plants in a hydrothermal interconnected system is considered: the authors use Lagrangian relaxation decomposition strategies, and a sequential quadratic programming algorithm to solve non-linear subproblems. Various Mixed Integer Linear Programming (MILP) approaches have been presented in the literature: for example, [37] and [38] used the Interior Point method within a Branch-and-Bound algorithm, while [11], [59], [60] and [39] used the Ilog-Cplex [71] MILP solver under GAMS.

Although we limit the analysis to the case of a single reservoir, the problem is especially interesting because both of its practical relevance and of the difficulties induced by its non-linear aspects, namely the relationship between the unit electrical power output and the corresponding water flow derived from the reservoir, particularly if the so called head effect, i.e., the influence on power production of the water level in the reservoir, has to be taken into account. We focus on the modeling of this non-linear characteristic, and show how it can be efficiently and accurately dealt with by using MILP techniques. Indeed, the high efficiency of modern MILP software tools, both in terms of solution accuracy and computing time, encourage their use also in the solution of non-linear problems.

The proposed MILP model allows one to accurately represent the main technical and operating characteristics of a pump-storage multi-unit hydro power plant, and turns out to be computationally solvable for a planning horizon of one week. For the general structure of the MILP model, we follow the one proposed in [39]. The differences mainly refer to the following aspects: (i) the proposed model takes into account some additional characteristics of the hydro units, such as ramp transition constraints and pump-storage operating mode; (ii) we introduce a more sophisticated modeling of the head effect through a specialized approximation methodology (based on two dimensional considerations) for the relationship among power, volume and flow.

The chapter is organized as follows. In Sections 6.2.1 and 6.2.2 we give the main components of the proposed model, with special emphasis on costs and constraints related to turbine and pump startups and to the linearization of the relationship between power and water flow. The most sophisticated version of the model, which allows a tight representation of the head effect, is presented in Section 6.3. The model is then computationally evaluated in Section 6.4 through experiments on real-world data. Instances where the pumps have prohibited zones are also evaluated. Conclusions and directions for future researches are finally given in Section 6.5.

6.2 Mathematical model

In our nomenclature all parameters are represented by upper capitals and all variables by lower capitals.

Preliminary observe that the parameters allow one to handle the pump start up in the two typical ways. If unit j is started up as a pump by using another turbine of the power plant, then there is no energy consumption, but the relevant water spillage is taken into account. The opposite holds if pump j is started up by using the energy provided by the external power network. In other words, the input has either $E_j = 0$ (in the former case) or $Y_j = 0$ (in the latter).

Note in addition that the first four variables are subject to the following obvious bounding constraints, for all $t \in T$ and $j \in J$:

$$\begin{array}{rcccl} Q_j^- & \leq & q_{jt} & \leq & \overline{Q}_j; \\ \underline{V} & \leq & v_t & \leq & \overline{V}; \\ P_j^- & \leq & p_{jt} & \leq & \overline{P}_j; \\ 0 & \leq & s_t & \leq & \overline{S}, \end{array}$$

and that, for any period t, the values of q_{jt} and p_{jt} depend on the three possible cases that can occur relative to turbine-pump unit j:

TP10: if unit j is generating power (i.e., $g_{jt} = 1$ and $u_{jt} = 0$) then both values are positive;

TP01: if unit j is pumping water (i.e., $g_{jt} = 0$ and $u_{jt} = 1$) then both values are negative;

TP00: if unit j is not operating (i.e., $g_{jt} = u_{jt} = 0$) then both values are zero.

The model we propose aims at maximizing the sum, over all periods, of the profit given by power selling, minus the start-up cost of each turbine-pump unit (if it occurs). Formally, this is represented by the linear objective function:

$$\max \sum_{j\in J}\sum_{t\in T}\Big(\Delta t\,\Pi_t\,p_{jt} - C_j\,\widetilde{w}_{jt} - (D_j + \Pi_t E_j)\widetilde{y}_{jt}\Big). \tag{6.1}$$

Note that the first term can take a negative value when the unit works as a pump.

The model can be logically subdivided into a set of "naturally" linear constraints and a set of non-linear constraints, that are linearized in order to handle the model through MILP techniques.

Section 6.2.1 reports the set of "naturally" linear constraints. Sections 6.2.2 and 6.3 are devoted to the treatment of the non-linear relationship between power production and water flow. In particular, we report in Section 6.2.2 an extension of the model of [39] and in Section 6.3 an enhanced version to better take into account the head effect.

6.2.1 Linear constraints

The relationships among flow, volume and pumps/turbines status can be modeled through the following linear constraints:

$$\begin{aligned}
v_{\bar{t}} - V_{\bar{t}} &= 0 && && (6.2)\\
v_t - v_{t-1} - 3600\,\Delta t\,(I_t - \sum_{j\in J} q_{jt} - s_t) &= 0 && \forall t \in T && (6.3)\\
q_{jt} - (Q_j^-\,u_{jt} + \underline{Q}_j\,g_{jt}) &\geq 0 && \forall j \in J, t \in T && (6.4)\\
q_{jt} - (Q_j^-\,u_{jt} + \overline{Q}_j\,g_{jt}) &\leq 0 && \forall j \in J, t \in T && (6.5)\\
\sum_{j\in J}(q_{jt} - q_{j(t-1)}) + \Delta q^- &\geq 0 && \forall t \in T && (6.6)\\
\sum_{j\in J}(q_{jt} - q_{j(t-1)}) - \Delta q^+ &\leq 0 && \forall t \in T && (6.7)\\
s_t - \sum_{j\in J}(W_j\,\widetilde{w}_{jt} + Y_j\,\widetilde{y}_{jt}) &\geq 0 && \forall t \in T && (6.8)\\
\sum_{j\in J} q_{jt} + s_t - \underline{\Theta} &\geq 0 && \forall t \in T && (6.9)\\
g_{jt} - g_{j(t-1)} - (\widetilde{w}_{jt} - w_{jt}) &= 0 && \forall j \in J, t \in T && (6.10)\\
\widetilde{w}_{jt} + w_{jt} &\leq 1 && \forall j \in J, t \in T && (6.11)\\
u_{jt} - u_{j(t-1)} - (\widetilde{y}_{jt} - y_{jt}) &= 0 && \forall j \in J, t \in T && (6.12)\\
\widetilde{y}_{jt} + y_{jt} &\leq 1 && \forall j \in J, t \in T && (6.13)\\
g_{jt} + u_{kt} &\leq 1 && \forall j, k \in J, t \in T && (6.14)\\
\sum_{j\in J} u_{jt} &\leq \overline{n} - 1 && \forall t \in T. && (6.15)
\end{aligned}$$

Constraint (6.2) sets the final water volume to the desired target value at the end of the considered time horizon $\bar{t}$. Constraints (6.3) impose the water conservation within two consecutive time periods. Constraints (6.4) and (6.5) establish lower and upper bounds on the flows in the turbines according to the three cases discussed above. Constraints (6.6) and (6.7) limit the flow variation within two consecutive periods. Constraints (6.8) impose the water spillage needed to startup a pump or a turbine. Constraints (6.9) establish a lower bound on the amount of water released in each period. Constraints (6.10) and (6.11) (resp. (6.12) and (6.13)) define the switch-on/switch-off rules of the turbines (resp. of the pumps). Constraints (6.14) impose that, if a turbine is on, no pump can be on and vice versa. Finally, constraints (6.15) are only introduced if the pumps startup method is to use the turbines: at least one pump is off because there are no turbines available to startup the last pump.

Note that an equivalent model could be obtained (see [34]) by eliminating the shutdown variables w_{jt} and y_{jt}, and replacing constraints (6.10)–(6.13) with:

$$g_{jt} - g_{j(t-1)} - \widetilde{w}_{jt} \leq 0 \ \forall j \in J, t \in T \tag{6.16}$$

$$u_{jt} - u_{j(t-1)} - \widetilde{y}_{jt} \leq 0 \ \forall j \in J, t \in T. \tag{6.17}$$

Indeed, the objective function (6.1) ensures that in any optimal solution the startup variables $\widetilde{w}_{jt}$ (resp. $\widetilde{y}_{jt}$) take the value 1 only if $g_{jt} - g_{j(t-1)} = 1$ (resp. $u_{jt} - u_{j(t-1)} = 1$). The resulting model is smaller, but this does not guarantee a better performance. Indeed, the two LP relaxations are identical, and the behavior of the MILP solver is unpredictable. As a matter of fact, for the benchmarks used in our experiments, the smaller model turned out to be equivalent to the larger one for the easy instances, but definitely worse for the difficult ones.

6.2.2 Linearizing the power production function

The performance of a hydro turbine depends on the rate of water discharge and on the net hydraulic head. The value of the net head depends on the water level in the reservoir, the tail-race level and the penstock losses (that are a function of the water flow). It follows that the power generated from a hydro unit is related to the water flow and the reservoir characteristics. For a generic hydro generator unit, the power output p can be expressed as a non-linear function φ of the water flow q and the water volume v in the reservoir, by including the non-linear relationship that links the net head value to the water volume and the water flow, as well as to the electric loss of the generator, i.e.,

$$p = \varphi(q, v). \tag{6.18}$$

(Note however that each unit will be characterized by a specific φ function.)

Even for a prefixed volume $\widetilde{v}$, the power production, as a function of the water flow, is non-linear and non-concave. Net head variation can only be ignored for relatively large reservoirs, in which case power generation is solely dependent on the water flow. (An example of (6.18) is provided in [101].)

An accurate approximation of φ is crucial for modeling the head effect. In [39] the function was approximated by considering a fixed number (three) of water volumes, say $\widetilde{v}^1$, $\widetilde{v}^2$, $\widetilde{v}^3$ and interpolating, for each $\widetilde{v}^r$, the resulting function

$$p = \varphi|_{\widetilde{v}^r}(q) \tag{6.19}$$

by piecewise linear approximation. To our knowledge, this has been the first successful modeling of the head effect. Indeed, a more accurate approximation of (6.18) through meshing and triangulation, proposed in [59], proved to be only suitable for small systems (see [60]).

We describe the improvement we propose for approximating (6.18) in two steps. In the present section we show how to (i) slightly generalize the approach in [39] to a parametric number of water volumes through a classical use of binary variables, and (ii) tighten the linear programming relaxation of the model through a more precise estimation of the upper bound on the power production. The second step, undertaken in the next section, introduces an accurate evaluation of the power production corresponding to intermediate water volumes.

While in [39] the piecewise linear approximation was formulated through the incremental method, we adopted the convex combination method which is mathematically equivalent (see, e.g., [75]) but allows a more intuitive explanation of the enhanced linearization that will be introduced in Section 6.3. We consider $\overline{r}$ volume intervals and $\overline{z}$ coordinates (*breakpoints*) along the flow axis. Let $R = \{1, \ldots, \overline{r}\}$ and $Z = \{1, \ldots, \overline{z}\}$. Let us introduce the following additional parameters:

$[H_{r-1}, H_r)$ = extreme water volumes for interval r $(r \in R)$ [m^3];

Q_{ji} = flow in turbine j at breakpoint i $(j \in J, i \in Z)$ [m^3/s];

P_{jir} = power from turbine j at breakpoint i for interval r $(j \in J, i \in Z, r \in R)$ [MW];

$\Delta P_{jr} = \max_{i \in Z}\{P_{ji\overline{r}} - P_{jir}\}$ $(j \in J, r \in R)$ [MW],

where the last value, introduced for ease of notation, represents the maximum power difference between intervals r and $\overline{r}$. Figure 6.1 depicts, for a given water volume, a classical power-flow characteristic of a turbine (dotted line) and its piecewise-linear approximation obtained with four breakpoints (solid line).

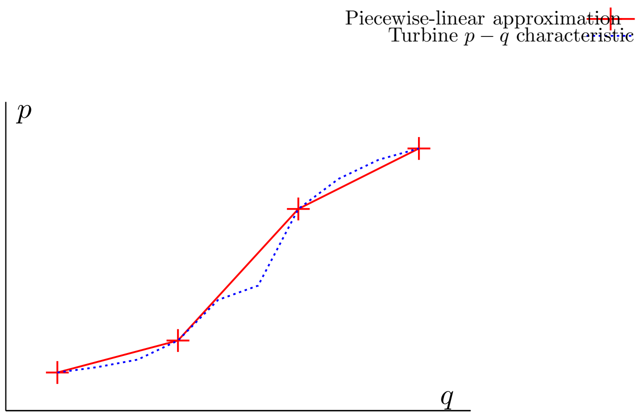


Figure 6.1: The simple approximation

Our linearization technique makes use of the following variables:

d_{tr} = membership status of volume v_t wrt interval r [1 if $H_{r-1} \leq v_t < H_r$, 0 otherwise] $(t \in T, r \in R)$;

z_{jti} = contiguity status of q_{jt} wrt to discretized flow Q_{ji} [1 if $Q_{j(i-1)} < q_{jt} \leq Q_{ji}$ or $Q_{ji} \leq q_{jt} < Q_{j(i+1)}$, 0 otherwise] ($j \in J, t \in T, i \in Z$);

λ_{jti} = weight of breakpoint i for turbine j in period t ($j \in J, t \in T, i \in Z$),

where the last variable must obey

$$0 \leq \lambda_{jti} \leq \quad 1.$$

The following constraints complete model (6.1)–(6.15) by approximating the power production function (6.18) with a parametric number of water volumes:

$$q_{jt} - \sum_{i \in Z} Q_{ji}\lambda_{jti} - Q_j^- u_{jt} = 0 \quad \forall j \in J, t \in T \tag{6.20}$$

$$\sum_{i \in Z} \lambda_{jti} - g_{jt} = 0 \quad \forall j \in J, t \in T \tag{6.21}$$

$$\lambda_{jti} - z_{jti} \leq 0 \quad \forall j \in J, t \in T, i \in Z \tag{6.22}$$

$$z_{jti} + z_{jtk} \leq 1 \quad \forall j \in J, t \in T, \forall i, k \in Z : i < k - 1 \tag{6.23}$$

$$\sum_{r \in R} d_{tr} = 1 \quad \forall t \in T \tag{6.24}$$

$$p_{jt} - \sum_{i \in Z} P_{jir}\lambda_{jti} - P_j^- u_{jt} - \Delta P_{jr}(1 - d_{tr}) \leq 0 \quad \forall j \in J, t \in T, r \in R \tag{6.25}$$

$$v_t - \sum_{r \in R} H_{r-1} d_{tr} \geq 0 \quad \forall t \in T \tag{6.26}$$

$$v_t - \sum_{r \in R} H_r d_{tr} \leq 0 \quad \forall t \in T. \tag{6.27}$$

Equations (6.20)–(6.23) express the water flow q_{jt} of turbine/pump j in period t in the three possible cases seen in Section 7.2. If $u_{jt} = 0$, the pump is off and the flow is either zero (if the turbine is off as well, case TP00) or a convex combination of breakpoint flows (case TP10); otherwise the pump is on (so $g_{jt} = 0$ from (6.14)) and there is a constant negative flow Q_j^- (case TP01). Observe indeed that constraints (6.22)–(6.23) are inactive when $g_{jt} = 0$ (due to (6.21)), so the first and third case are directly modeled by (6.20). If instead $g_{jt} = 1$ constraints (6.21) impose that the breakpoint weights sum up to one. Due to constraints (6.22), any λ_{jti} can only be non-zero if the corresponding binary variable z_{jti} is one. It follows that constraints (6.22) and (6.23) together ensure that: (i) at most two weights can take a positive value, and (ii) if this occurs for exactly two weights then they must be contiguous. In summary, the overall effect of constraints (6.20)–(6.23) is that q_{jt} is either a constant negative value Q_j^- (case TP01), or null (case TP00), or a piecewise linear approximation (case TP10).

Similarly, equations (6.24)–(6.25) express the power p_{jt} of turbine/pump j in period t for volume interval r, in the same three cases. Due to (6.24), all d_{tr}'s are zero but one. Hence, in the unique volume interval, say $\widetilde{r}$, for which $d_{t\widetilde{r}} = 1$, the last term of (6.25) takes the value 0 and (6.25) itself assumes the same form as (6.20), but with powers instead of flows. It follows that the same considerations used above ensure that equation (6.25) model the three possible cases. The only difference is in the '$\leq$' sign, which is adopted here, instead of '$=$'. However: (i) this has no effect on the power production, since the objective function (6.1) ensures the constraint tightness; (ii) for all the other volume intervals $r \neq \widetilde{r}$, for which $d_{tr} = 0$, the last term of (6.25) takes the value ΔP_{jr}, thus deactivating the constraint. Finally, equations (6.26)–(6.27) define, for each time period t, the two extreme water volumes of the interval where the computed volume v_t lies.

Constraints (6.25) are the crucial difference with respect to the way (6.18) is approximated in [39]:

(i) it is obvious that increasing the number of volume intervals improves the approximation. However, as shown in Section 6.4, this number cannot be increased too much in practice without making the model too big to be handled within reasonable CPU times;

(ii) constraints (6.25) could be de-activated (as in [39]) by using the overestimated constant value $\overline{P}_j$ instead of ΔP_{jr}. It is known however that adopting tighter values highly strengthens the linear programming relaxation (as confirmed by the computational experiments of Section 6.4).

6.3 Enhancing the linearization

In the model of the previous section, for any volume v_t belonging, say, to the r-th interval $[H_{r-1}, H_r)$, the power production is approximated through a prefixed (static) value P_{jir} depending on the turbine and the breakpoint (see (6.25)). The accuracy obtainable in this way heavily depends on the number $\overline{r}+1$ of water volumes H_r (corresponding to $\overline{r}$ volume intervals). Such a number, however, cannot be too high without substantially affecting the computational effort. We next show how a good approximation can be obtained by keeping $\overline{r}$ at an effective low value by introducing an enhanced linearization that corrects the estimated power production through two dimensional considerations.

In the enhanced model, for a volume v_t, belonging, say, to interval $[H_{r-1}, H_r)$, instead of approximating the power production by selecting a point on a single piecewise linear function, we approximate it through a weighted combination of values computed for the two extremes H_{r-1} and H_r. Let us introduce the power excursion, for turbine j and breakpoint i, between intervals r and $r+1$,

$$\Delta P_{jir} = P_{ji(r+1)} - P_{jir} \ (j \in J, i \in Z, r \in R) \ \text{[MW]},$$

and the quantity

$$\overline{\Delta P}_{jir} = \max_{k \in Z}\{P_{jk(r+1)} - P_{jkr}\} - \Delta P_{jir} \ (j \in J, i \in Z, r \in R) \ \text{[MW]}$$

(with $P_{ji(\overline{r}+1)} = P_{ji\overline{r}}$), which is used to de-activate constraints, as will be shown below. Moreover, the enhanced model requires the additional variables

$$d'_{tr} = d_{tr}(v_t - H_{r-1})/(H_r - H_{r-1}) \ (t \in T, r \in R)$$

to represent the weight used for combining the values computed for the two extremes of volume interval r.

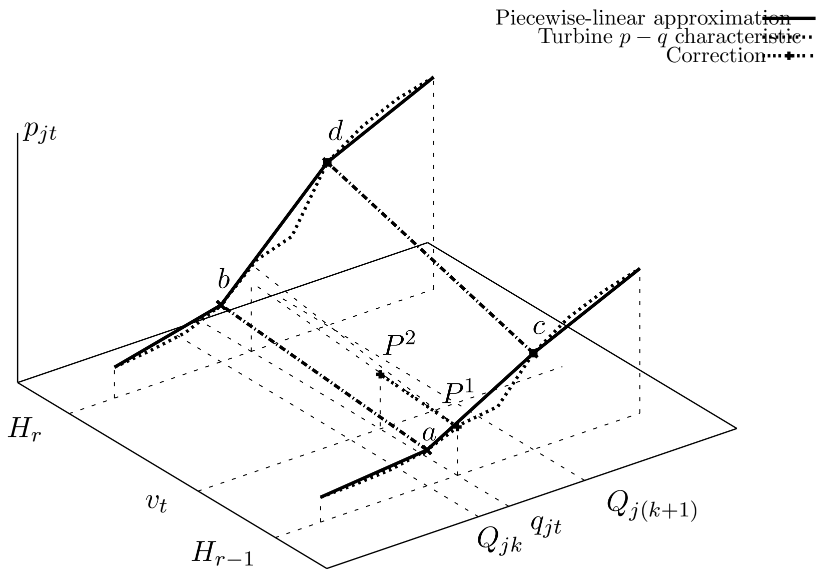


Figure 6.2: The enhanced approximation

The enhanced linearized power production function is then computed by equations (6.20)–(6.24), and:

$$d'_{tr} - d_{tr} \le 0 \;\; \forall t \in T, r \in R \tag{6.28}$$

$$p_{jt} - P^-_j u_{jt} - \overline{P}_j g_{jt} \le 0 \;\; \forall j \in J, t \in T \tag{6.29}$$

$$p_{jt} - \left(\sum_{k \in Z} P_{jkr} \lambda_{jtk} + \Delta P_{jir} d'_{tr} \right) - \Bigg(\Delta P_{jr}(1 - d_{tr}) + \overline{\Delta P}_{jir}(1 - z_{jti}) \Bigg) \le 0 \forall j \in J, t \in T, r \in R, i \in Z \tag{6.30}$$

$$v_t - \sum_{r \in R} \left(H_{r-1} d_{tr} + \; (H_r - H_{r-1}) d'_{tr} \right) = 0 \forall t \in T, \tag{6.31}$$

which replace (6.25)–(6.27).

Equations (6.28) ensure that, for any time period t, the only non-zero weight d'_{tr} can occur for the unique interval r for which $d_{tr} = 1$ (see (6.24)). As a consequence, in the summation of equations (6.31) the only non-zero term must be equal to the value of v_t (given by (6.3)), thus uniquely determining the corresponding value $d'_{tr} = (v_t - H_{r-1})/(H_r - H_{r-1})$.

Equations (6.29) are only active when turbine j is off in period t. They thus define the (negative) power consumption due to pump j (case TP01 of Section 7.2), possibly equal to zero if pump j is off as well (case TP00).

Similarly, equations (6.30) are only active when turbine j is on in period t (case TP10), since otherwise equations (6.29) impose a negative upper bound P^-_j on p_{jt}, hence dominating any non-negative upper bound produced by (6.30). When active, they determine the (positive) upper bound on the power production, tighter than the one imposed by (6.29), which is just the variable upper bound $\overline{P}_j$. As previously observed, p_{jt} will exactly match such a bound, since the objective function (6.1) maximizes the power production. Note that in case TP10 the two terms within brackets of (6.30) play the same role as the second and fourth term of (6.25), respectively. More precisely, $\Delta P_{jir} d'_{tr}$ imposes a correction to the regular term $\sum_{k \in Z} P_{jkr}\lambda_{jtk}$ while $\overline{\Delta P}_{jir}(1 - z_{jti})$ is used to deactivate those constraints (6.30) whose breakpoint i is not used (i.e., those for which $z_{jti} = 0$).

The correction $\Delta P_{jir} d'_{tr}$ is depicted in Figure 6.2. Since $v_t \in [H_{r-1}, H_r)$, the power approximation obtained by the model of the previous section would be $P_{jkr}\lambda_{jtk} + P_{j(k+1)r}\lambda_{jt(k+1)}$ (with $\lambda_{jt(k+1)} = 1 - \lambda_{jtk}$), i.e., the value P^1 in the figure. The enhancement given by equations (6.28)–(6.31) produces a better approximation, namely value P^2 in the figure. Indeed the correction parameter d'_{tr} is computed by considering the relative position of v_t within the volume interval $[H_{r-1}, H_r)$ (see equation (6.31)). Note that two constraints (6.30) are active at the same time: the one for k and the one for $k + 1$. However, to avoid an overestimation of the correction, the tighter constraint is the one with the smallest ΔP_{jir} value. In Figure 6.2 such value is ΔP_{jkr} , since the slope of segment ab is smaller than that of segment cd.

We illustrate the enhanced linearization technique through a numerical example. Consider Figure 6.2, and assume that the (q, v, p) coordinates of the interested points are: $a = (18, 1, 22)$, $b = (18, 5, 34)$, $c = (28, 1, 58)$ and $d = (28, 5, 98)$. Let us compute the power production corresponding to $q_{jt} = 20$ and $v_t = 2.5$. The linearization of Section 6.2.2 would give $P^1 = 22\lambda_{jtk} + 58\lambda_{jt(k+1)} = 29.2$ (by equation (6.25) with $\lambda_{jtk} = 0.8$). The enhanced linearization gives $P^2 = P^1 + (34 - 22)d'_{tr} = 33.7$ (by equation (6.30) with $d'_{tr} = 0.375$).

6.4 Computational Results

The models presented in the previous sections were tested by running the MILP solver Ilog-Cplex 10.0 [71] under mathematical programming modeling language AMPL Version 20061102. The tests were executed by sequentially running the code on a single processor of an Intel Core2 CPU 6600, 2.40 GHz, 1.94 GB of RAM. For each instance, a time limit of 7,200 seconds was imposed.

Three real-word instances were considered referring to a hydro power plant with one Francis turbine fed by a reservoir of capacity $33 \cdot 10^6$ m^3, with a maximum level of 85.25 m. We considered the water inflows and electricity market prices of a week of three different months (namely, April, June and December), selected so as to have considerably different scenarios, with hourly time periods (168 periods per instance). The complete instances are available on line at `http://www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm`.

We give the results obtained for these instances by three models:

(i) a basic model, without the improvements introduced in Section 6.2.2('BDLM-' in the tables), mimicking the model in [39];

(ii) the improved model of Section 6.2.2 ('BDLM' in the tables);

(iii) the final enhanced model of Section 6.3 ('BDLM+' in the tables). The models were tested with two different φ functions, called φ_1 and φ_2, for the turbine (see (6.18)). Each power production function is approximated by considering three fixed values H_i of the water volume. For each water volume, the $p - q$ relationship (6.19) is represented by a piecewise linear approximation with five breakpoints, as shown in Figure 6.3. As already mentioned, in pumping operating mode we assume a constant water flow value (of 0.64 pu) as well as a constant power consumption (of 0.85 pu).

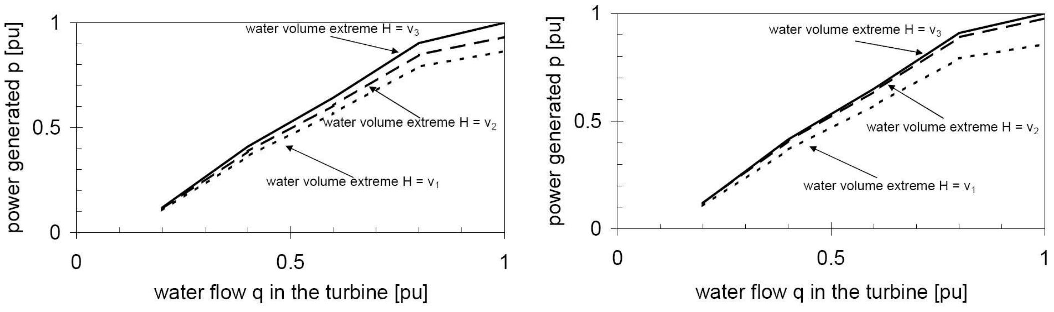


Figure 6.3: Piecewise approximation of the relationship (6.19) for three volume values

The results for φ_1 and φ_2 are given in Tables 6.1 and 6.2, respectively. For each instance and model the entries give:

(a) the total number of variables, the number of binary variables, and the number of constraints after Ilog-Cplex preprocessing;

(b) the value of the initial LP relaxation, obtained by replacing each binary constraint of type $x_i \in \{0, 1\}$ with $0 \leq x_i \leq 1$;

(c) the value of the improved LP relaxation computed by Ilog-Cplex at the root node through its default cutting plane separation;

(d) the value of the best solution computed by Ilog-Cplex within the time limit;

(e) the final percentage gap, computed as

$$100 \cdot \frac{\text{Best upper bound} - \text{Best solution value}}{\text{Best solution value}};$$

(f) the number of Branch-and-Bound nodes;

(g) the number of unsolved nodes when the time limit is reached;

(h) the total CPU time spent.

The first two lines of each instance refer to two models (BDLM- and BDLM) that provide the same level of approximation of the system (note indeed that the best solution values are identical). By comparing them we can observe that both the initial and the improved LP relaxation of BDLM are much tighter than those of BDLM-. Numbers of nodes and CPU times are generally competitive or much lower for BDLM. Both models produced the optimal solution for all instances within very short CPU times. The difference in the behavior of the two models is mainly due to the use of the tighter ΔP_{jr} values in (6.25).

The third line of each instance refer to our most sophisticated model (BDLM+), which provides a more accurate approximation of the real system. This is confirmed by the computational results, which show a considerably better solution value. The higher complexity of this model is also reflected by the larger computing times. In spite of this, five instances out of six were solved to optimality. Moreover, for the two 'hard' instances of Table 6.1,

Table 6.1: Results for a turbine with the φ_1 characteristic of Figure 6.3

Instance	Model	# of vars	# of bin vars	# of const	Initial LP relaxation value	Improved LP relaxation value	Best solution value	Final % gap	Number of nodes	Number of unsolved nodes	CPU time
	BDLM-	3,677	2,166	6,020	158,802.93	47,870.30	30,371.44	0.00	489	0	4.25
`April_T168`	BDLM	3,836	2,325	5,517	37,725.41	33,651.66	30,371.44	0.00	400	0	3.57
	BDLM+	4,325	2,327	8,191	95,603.39	46,597.83	42,298.27	0.00	3,688	0	24.16
	BDLM-	3,668	2,157	5,995	344,944.69	180,672.99	125,858.46	0.00	21,007	0	91.53
`June_T168`	BDLM	3,856	2,345	5,541	135,425.69	127,511.18	125,858.46	0.00	10,551	0	29.10
	BDLM+	4,325	2,347	8,227	175,210.15	147,520.10	143,688.22	0.00	572,824	0	2,555.67
	BDLM-	3,693	2,182	6,045	392,563.70	196,000.43	154,702.66	0.00	2,336	0	14.42
`December_T168`	BDLM	3,852	2,341	5,541	167,127.86	158,217.74	154,702.66	0.00	2,834	0	10.47
	BDLM+	4,349	2,343	8,223	222,895.80	185,008.07	176,519.74	0.69	1,521,821	816,717	7,492.94

Table 6.2: Results for a turbine with the φ_2 characteristic of Figure 6.3

Instance	Model	# of vars	# of bin vars	# of const	Initial LP relaxation value	Improved LP relaxation value	Best solution value	Final % gap	Number of nodes	Number of unsolved nodes	CPU time
	BDLM-	3,677	2,166	6,020	160,254.15	48,111.81	31,098.94	0.00	784	0	6.86
`April_T168`	BDLM	3,836	2,325	5,517	38,342.76	35,779.80	31,098.94	0.00	687	0	4.03
	BDLM+	4,325	2,327	8,191	87,151.83	40,533.52	38,347.13	0.00	3,890	0	23.77
	BDLM-	3,668	2,157	5,995	347,101.06	180,389.17	132,930.96	0.00	5,814	0	32.34
`June_T168`	BDLM	3,856	2,345	5,541	136,061.14	133,282.18	132,930.96	0.00	2,472	0	13.02
	BDLM+	4,325	2,347	8,227	162,377.67	136,725.37	135,030.48	0.00	11,507	0	62.88
	BDLM-	3,693	2,182	6,045	394,369.19	202,374.20	161,954.09	0.00	1,634	0	9.40
`December_T168`	BDLM	3,852	2,341	5,541	170,012.29	165,791.65	161,954.09	0.00	2,001	0	8.03
	BDLM+	4,349	2,343	8,223	208,740.36	172,898.97	169,283.99	0.00	20,549	0	99.19

June_T168 and December_T168, the incumbent solution values after 300 CPU seconds were already 143,619.22 and 176,508.30, respectively, i.e., very close to the best solution values.

In Table 6.3 we examine the size of the three models by giving the number of variables and constraints in terms of the main parameters, namely ($\bar{t}$, $\bar{r}$ and $\bar{z}$). The percentage of binary variables was, for all cases, between 50 and 60 % of the total.

Table 6.4 shows, for instance April_T168 of Table 6.1, how the level of approximation of BDLM improves with the number $\bar{r}$ of volume intervals. The best solution value considerably improves when $\bar{r}$ is increased to 4 and 5, by still requiring acceptable CPU times. Going to higher values (7 and 10) the improvement is marginal, while the increased number of variables and constraints makes the computational effort very heavy, and the model is not solved to optimality within the time limit. In any case, the best solution value remains far from the one produced by BDLM+ with $\bar{r} = 3$ (shown in the last line of the table).

In order to further compare the solutions achieved by models BDLM and BDLM+ we computed the value of the objective function obtained in model BDLM+ when the solution attained by the BDLM model is enforced. We considered a turbine with function φ_1 (see Figure 6.3). The three columns of Table 7.6 give the value of the solution produced by BDLM, the value produced by BDLM+ if the values of all variables in the BDLM solution (except p_{jt}) are enforced, and the value produced by BDLM+ from scratch. The results show that the enhanced linearization of BDLM+ allows the MILP solver to find truly improved solutions wrt BDLM, i.e., the different values in the tables do not merely correspond to different measures.

Other authors considered real cases in which the turbines have forbidden operating zones. For example, [117] considers one multi-plant daily instance (24 time periods) in which each plant has a single forbidden flow interval. In order to test our model on such situations, we extracted from such instance three single plant instances (named S1, S2 and S3), and added to the BDLM+ model specific constraints to forbid a flow interval, namely:

$$q_{jt} - \underline{F}_j - \overline{Q}_j(1 - x_{jt}) \leq \; 0 \quad \forall j \in J, t \in T \tag{6.32}$$

$$q_{jt} - \overline{F}_j(1 - x_{jt}) - Q_j^- x_{jt} \geq \; 0 \quad \forall j \in J, t \in T, \tag{6.33}$$

where $[\underline{F}_j, \overline{F}_j]$ is the prohibited operating zone for turbine j, and x_{jt} is a binary variable that takes the value 1 when $Q_j^- \leq q_{jt} \leq \underline{F}_j$ or the value 0 when $\overline{F}_j \leq q_{jt} \leq \overline{Q}_j$. Note that (6.32) (resp. (6.33)) is not active when $x_{jt} = 0$ (resp. $x_{jt} = 1$). The outcome of the experiments is reported in Table 6.4, where the second column tells whether constraints (6.32)-(6.33) are added to BDLM+. The first two lines of each instance give the results obtained by solving such instances with BDLM+, with and without the new constraints. The optimal solutions only slightly differ on instance S1, thus showing that (6.32)-(6.33) were (almost) not active. In order to better test the model, we artificially moved the forbidden intervals to optimal regions, thus creating more challenging instances S1A, S2A and S3A. The behavior of BDLM+ with (6.32)-(6.33), given in the third line of each instance, results to be still satisfactory.

In order to illustrate the differences between the two proposed models, BDLM and BDLM+, Figures 6.4, 6.5, 6.6 and 6.7 show the scheduling results they produce for instance June_T168 with power production function φ_1.

Figure 6.4 shows the water volume values in the basin, taking into account the minor amount of natural inflow (just fractions of m^3/s) and the different volume values at the beginning and at the end of the week, imposed by the considered instance (namely $32.5 \cdot 10^6$ m^3 and $26.5 \cdot 10^6$ m^3, respectively). The horizontal lines show the three volume intervals

Table 6.3: Number of variables and constraints for the three models considering 8 configurations of ($\bar{t}$; $\bar{r}$; $\bar{z}$)

Model	(24;3;5)	(24;3;8)	(24;5;5)	(24;5;8)	(168;3;5)	(168;3;8)	(168;5;5)	(168;5;8)
BDLM-	528; 937	672; 18,217	528; 937	672; 18,217	3,696; 6,553	4,704; 127,513	3,696; 6,553	4,704; 127,513
BDLM	552; 745	696; 18,025	600; 793	744; 18,073	3,864; 5,209	4,872; 126,169	4,200; 5,545	5,208; 126,505
BDLM+	648; 1,105	792; 18,529	744; 1,393	888; 18,961	4,536; 7,729	5,544; 129,697	5,208; 9,745	6,216; 132,721

Table 6.4: Results with more volume intervals for `April_T168` and a turbine with the characteristic of Figure 6.3

Model	Numer of volume intervals	# of vars	# of bin vars	# of const	Best solution value	Final %gap	Number of nodes	Number of unsolved nodes	CPU time
BDLM	3	3,836	2,325	5,517	30,371.44	0.00	400	0	3.57
	4	4,011	2,500	5,709	32,985.90	0.00	21,411	0	65.14
	5	4,170	2,659	5,877	34,299.61	0.00	30,537	0	88.63
	7	4,499	2,988	6,216	34,721.34	4.11	2,316,701	1,458,840	7,554.98
	10	4,986	3,475	6,720	35,577.54	8.11	1,589,801	996,593	7,525.20
BDLM+	-	4,325	2,327	8,191	42,298.27	0.00	3,688	0	24.16

adopted for linearizing the power production function. As already mentioned, the inflow and the initial and the final values were taken from the real world data of a power plant. Figure 6.5 compares the calculated profiles of the water flows discharged or pumped by the hydro unit during the considered week.

Figure 6.6 compares the power production levels, also giving the considered market price profile during the week, while Figure 6.7 compares the curves of the accumulated profits in the two models. Figure 6.6 shows that the maximum output of the unit calculated by BDLM+ is larger than that obtained by BDLM, due to the more refined representation of the head effect in function φ, that results in a higher profile of the calculated water volumes in the basin.

Table 6.5: Results for BDLM+ with and without the BDLM solution enforced

Instance	BDLM	BDLM+ with BDLM enforced	BDLM+
`April_T168`	30,371.44	41,651.54	42,298.27
`June_T168`	125,858.46	143,384.79	143,688.22
`December_T168`	154,702.66	175,339.08	176,519.74

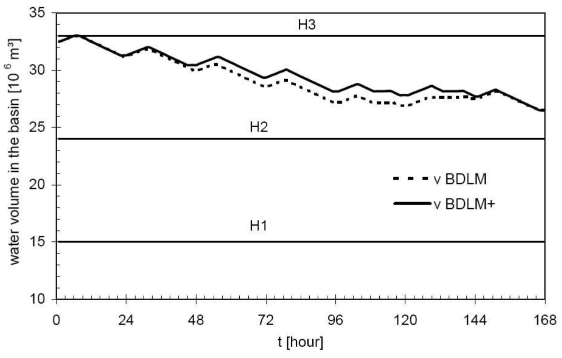


Figure 6.4: Water volumes

This explains (i) the larger profit levels estimated by BDLM+, as shown in Figure 6.7, and (ii) the different scheduling, mainly for the second last day (Sunday), characterized by lower market prices. (The considered week starts on Monday midnight and runs through the next Monday midnight.) For Sunday morning, BDLM suggests not to produce, due to the low value of the water volume stored in the basin (which should be saved in order to be available at the more profitable market price levels of the following day), whilst BDLM+ recommends to produce, allowing therefore a superior exploitation of the natural resource.

6.5 Conclusions

We have considered the problem of determining the commitment and the power generation of a single reservoir pump-storage hydro power plant. Starting from the MILP model proposed in [39], we have obtained an enhanced model that takes into account relevant technological aspects, such as ramp transitions, pump-storage and head effect. In particular, we have proposed a sophisticated approximation of the head effect in which the linearization is enhanced through two dimensional considerations. The proposed MILP model allows to accurately

Table 6.6: Results for the MILP model with 7 volume intervals and 5 breakpoints

Instance	(6.32)-(6.33) imposed	Initial LP relaxation value	Improved LP relaxation value	Best solution value	# nodes	CPU Time
S1	no	146,621.07	141,115.26	138,147.38	13,940	14.12
S1	yes	146,621.07	141,097.97	138,144.67	13,403	14.99
S1A	yes	146,621.07	141,002.27	137,540.30	2,535,628	2,507.13
S2	no	158,510.03	152,301.95	134,148.57	64,798	76.07
S2	yes	158,510.03	151,532.19	134,148.57	66,553	74.85
S2A	yes	158,510.03	152,117.09	134,141.57	57,645	72.10
S3	no	251,528.59	229,658.19	199,610.76	4,428	8.76
S3	yes	251,528.59	230,124.38	199,610.76	6,381	13.73
S3A	yes	251,528.59	229,359.16	199,277.13	3,456	8.58

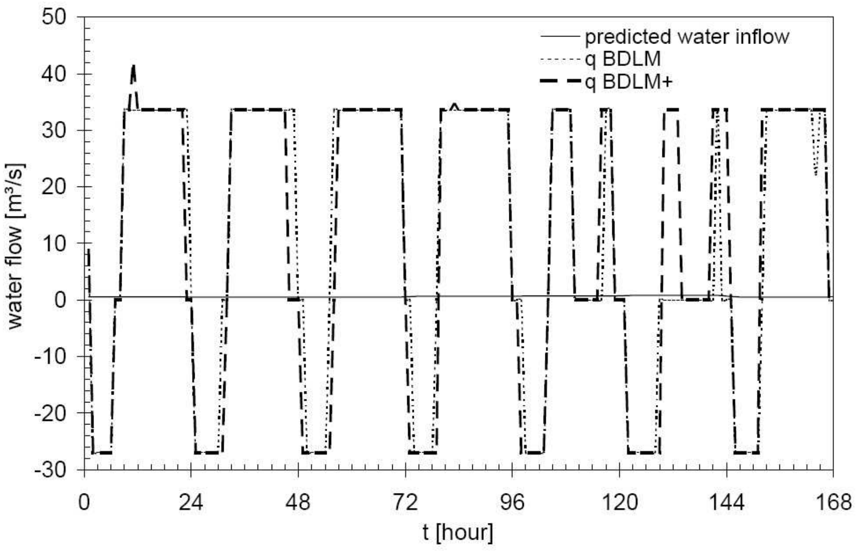


Figure 6.5: Inflow and flows

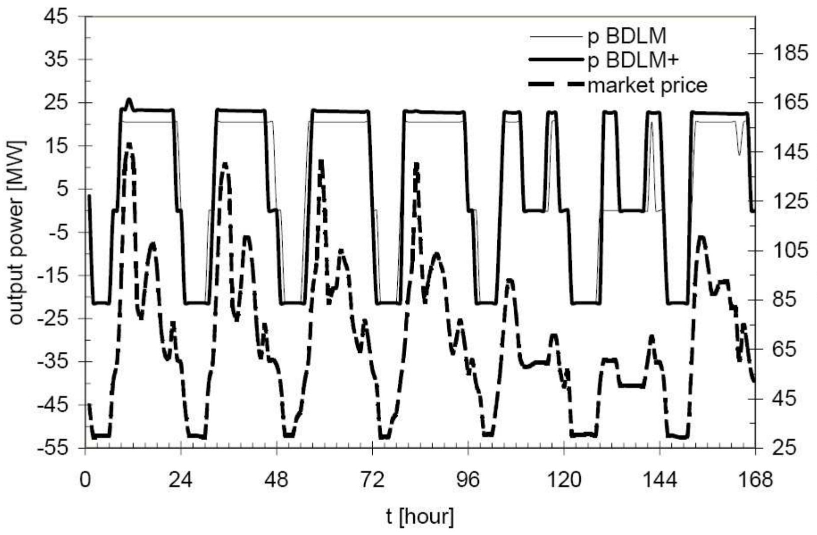


Figure 6.6: Price and powers

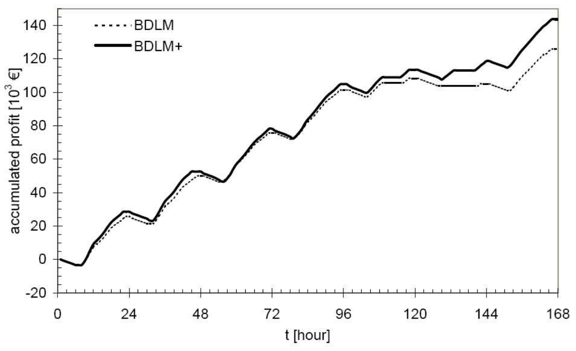


Figure 6.7: Profit

represent most of the hydro-electric system characteristics, and turns out to be computationally solvable for a planning horizon of one week, proving the high efficiency of modern MILP software tools, both in terms of solution accuracy and computing time.

Future developments could involve the extension of the model to represent hydrological interdependent plants in cascade hydro systems. This task is far from being trivial for various reasons. It is clear that the model would require the continuity equations of the hydro reservoirs, taking into account the relevant constraints such as branch flow limits and water travel time (see, e.g., [37, 38, 59, 39, 49]). The main drawback, however, could be that a cascade hydro system model for, say, k power plants would multiply by k the number of variables and constraints of BDLM+, and the computational experiments of Section 6.4 have shown that the performance of the model is heavily affected by its size. In addition, the sophisticated linear approximations introduced to model the non-linear aspects could deteriorate the accuracy of a much larger model. In our opinion the extension to the modeling of schemes with a combination of multiple reservoirs, rivers, weirs and hydro-electric plants in series and parallel combinations could require a different use of the MILP solver, which should not be run as a black box, but embedded in a more involved algorithmic framework.

6.6 Acknowledgments

The authors thank Professor Carlo Alberto Nucci for his helpful comments. We also thank anonymous referees for useful comments that improved the presentation.

Chapter 7

Water Network Design Problem

[1]

Introduction

The optimal design of a WDN (Water Distribution Network) consists, in its classical formulation, of the choice of a diameter for each pipe, while other design properties are considered to be fixed (e.g., the topology and pipe lengths). From a mathematical viewpoint, we can cast the optimal design problem of a WDN as an MINLP (Mixed Integer Non-Linear Programming) problem in which the discrete variables select from a set of commercially-available diameters, water flows and pressures must respect the hydraulic constraints, and we seek to minimize the cost function which only depends on the selected diameters.

Recently there has been renewed interest in optimal WDN design, due to emerging issues related to water distribution systems; in particular, the gradual deterioration of network pipes and the need for a more rational use of water resources has led to very costly renovation activities.

Approaches in the literature use various combinations of linearization and relaxation, which lead to MILP (Mixed Integer Linear Programming), NLP (Non-Linear Programming) and meta-heuristic algorithms. We survey these approaches in Section 7.4. In this chapter we are interested in approaches exploiting mathematical-programming formulations, and we consider two cases.

The MILP approach to our problem relies on using piecewise-linear approximations. If tractable, a solution of such a model would provide a global optimum of an approximation to the real system. If accurate models are desired for a large network, we are led to using a large number of binary variables (to manage the linear pieces). This tends to lead to a very poor relaxation and ultimately an intractable model.

With an MINLP approach, we are led to a more natural model. Our view is that by accurately modeling the non-linear phenomena, we will have a model that will provide an MINLP search with a good NLP relaxation. While foregoing any hope of practically verifying

[1]The results of this chapter appears in: C. Bragalli, C. D'Ambrosio, J. Lee, A. Lodi, P. Toth, Water Network Design by MINLP, IBM Research Report RC24495, 02/2008 (under revision).

MINLP global optimality of the best solution obtained, we are able to find very good solutions to large real-world instances.

Our experiments were carried out using AMPL ([55]) as an interface to MINLP codes. In a preliminary version of this work [30], we used Sven Leyffer's code `MINLP_BB` ([82], available from the University of Dundee) as well as the — at that time new — CMU/IBM open-source MINLP code `Bonmin v. 0.1` ([20, 24]), available from COIN-OR. In fact, it was in the context of our investigations that `Bonmin` was adapted for use on non-convex MINLP problems.

Our modeling and solution methods are worked out with the target software in mind (in particular, the Branch-and-Bound implementation in `Bonmin v. 0.1`), and the improved results on this full version of the chapter are all obtained by using `Bonmin v. trunk` (the development version), i.e., by implementing our special features in the development version of the code. We note that the open-source nature of `Bonmin` enabled us to rapidly test our ideas and then make them available to the developers and the users of `Bonmin` under the same open-source license used by `Bonmin` (Common Public License Version 1.0 (CPL)).

In Section 7.1, we formally set notation for specifying instances of the problem. In Section 7.2, we describe the problem more fully, through a preliminary continuous model and we discuss the two main modeling contributions of the chapter, namely a continuous objective function (see, Section 7.3) and a smooth (approximate) relaxation of the pressure loss in water pipes (see, Section 7.3.1). In Section 7.4.1, we survey earlier approaches, while in Section 7.4.2 we describe how we incorporate binary variables for the purposes of then applying MINLP codes. In Section 7.4.3, so as to decrease the non-linearity and non-convexity, we describe a reparameterization of pipes dimension by (cross-sectional) area, rather than diameter. In Section 7.5, we describe the results of computational experiments. Finally, in Section 7.6 we draw some conclusions.

7.1 Notation

The network is oriented for the sake of making a careful formulation, but flow on each pipe is not constrained in sign (i.e., it can be in either direction). The network consists of pipes (arcs) and junctions (nodes). In the optimization, the pipes are to have their diameters sized at minimum cost.

Sets:

E = set of pipes.

N = set of junctions.

S = set of source junctions (also called reservoirs, $S \subset N$).

$\delta_+(i)$ = set of pipes with tail at junction i ($i \in N$).

$\delta_-(i)$ = set of pipes with head at junction i ($i \in N$).

Parameters:

$len(e)$ = length of pipe e ($e \in E$).

$k(e)$ = physical constant depending on the roughness of pipe e ($e \in E$).

$d_{min}(e)$ = minimum diameter of pipe e ($e \in E$).

$d_{max}(e)$ = maximum diameter of pipe e ($e \in E$).

$v_{max}(e)$ = maximum speed of water in pipe e ($e \in E$).

$dem(i)$ = demand at junction i ($i \in N \setminus S$).

$elev(i)$ = physical elevation of junction i ($i \in N \setminus S$).

$ph_{min}(i)$ = minimum pressure head at junction i ($i \in N \setminus S$).

$ph_{max}(i)$ = maximum pressure head at junction i ($i \in N \setminus S$).

$h_s(i)$ = fixed hydraulic head of source junction i ($i \in S$).

For each pipe e, the available diameters belong to a discrete set of r_e elements. For $e \in E$:

$$d_{min}(e) := \mathcal{D}(e,1) < \mathcal{D}(e,2) < \cdots < \mathcal{D}(e,r_e) =: d_{max}(e) \ .$$

For each pipe $e \in E$, there is a cost function $C_e()$ having a discrete specification as a (typically rapidly) increasing function of diameter. That is, $\mathcal{C}(e,r) := C_e(\mathcal{D}(e,r))$, $r = 1,\ldots,r_e$, where:

$$\mathcal{C}(e,1) < \mathcal{C}(e,2) < \cdots < \mathcal{C}(e,r_e) \ .$$

7.2 A preliminary continuous model

In this section, we describe the problem, and at the same time we develop a preliminary NLP relaxation. Our goal is to develop a smooth NLP formulation that accurately models the problem.

Variables:

$Q(e)$ = flow in pipe e ($\forall\ e \in E$).

$D(e)$ = diameter of pipe e ($\forall\ e \in E$).

$H(i)$ = hydraulic head of junction i ($\forall\ i \in N$).

Simple bounds [Linear]:

$d_{min}(e) \leq D(e) \leq d_{max}(e) \quad (\forall\ e \in E)$.

$ph_{min}(i) + elev(i) \leq H(i) \leq ph_{max}(i) + elev(i) \quad (\forall\ i \in N \setminus S)$.

$H(i) = h_s(i) \quad (\forall\ i \in S)$.

The hydraulic head is the total energy per unit of weight of the water, and it is expressed in terms of a height. Furthermore, the hydraulic head is the sum of pressure head (ph), elevation head ($elev$) and velocity head ($\frac{v^2}{2g}$), all of which are measured in units of length. Velocity head (kinetic energy) is usually ignored because is much smaller than the elevation and pressure head (see [125]).

Flow bounds (dependent on cross-sectional area of pipe) [Smooth but non-convex]:

$$-\tfrac{\pi}{4}v_{max}(e)D^2(e) \le Q(e) \le \tfrac{\pi}{4}v_{max}(e)D^2(e) \quad (\forall\ e \in E).$$

Flow conservation [Linear]:

$$\textstyle\sum_{e\in\delta_-(i)} Q(e) - \sum_{e\in\delta_+(i)} Q(e) = dem(i) \quad (\forall\ i \in N \setminus S).$$

Head loss across links [Nonsmooth and non-convex]:

$$H(i)-H(j) = \mathrm{sgn}(Q(e))|Q(e)|^{1.852} \cdot 10.7 \cdot len(e) \cdot k(e)^{-1.852}/D(e)^{4.87} \quad (\forall\ e = (i,j) \in E).$$

This last constraint models pressure loss in water pipes due to friction using the empirical *Hazen-Williams equation.* This is an accepted model for fully turbulent flow in *water* networks (see [124]). Diameter is bounded away from 0, so the only nondifferentiability is when the flow is 0.

Such a nondifferentiability is discussed in details in Section 7.3.1.

Objective to be minimized [Discrete]:

$$\textstyle\sum_{e\in E} C_e(D(e))\ len(e)\ .$$

Because we only have discretized cost data, within AMPL we are fitting a polynomial to the input discrete cost data to make a *smooth* working continuous cost function $C_e()$.

Our motivation for that is to use a smooth function to closely fit the discrete cost data and the details of such a choice together with the relationship between continuous and discrete objective functions are discussed in the following section. In addition, computational experiments comparing the two options are reported at the end of Section 7.5.2.

7.3 Objective function

We have experimented with different fits: l_1, l_2 and l_∞; with and without requiring that the fit under or over approximates the discrete points. Requiring an under approximation makes our formulation a true relaxation — in the sense that the global minimum of our relaxation is a lower bound on the discrete optimum. We use and advocate weighted fits to minimize relative error. For example, our least-squares fit for pipe e minimizes

$$\sum_{r=1}^{r_e} \frac{\left[\mathcal{C}(e,r) - \left(\sum_{j=0}^{t_e} \beta(j,e)\left(\frac{\pi}{4}\mathcal{D}(e,r)^2\right)^j\right)\right]^2}{\mathcal{C}(e,r)^2} = \sum_{r=1}^{r_e}\left[1 - \left(\frac{\sum_{j=0}^{t_e} \beta(j,e)\left(\frac{\pi}{4}\mathcal{D}(e,r)^2\right)^j}{\mathcal{C}(e,r)}\right)\right]^2,$$

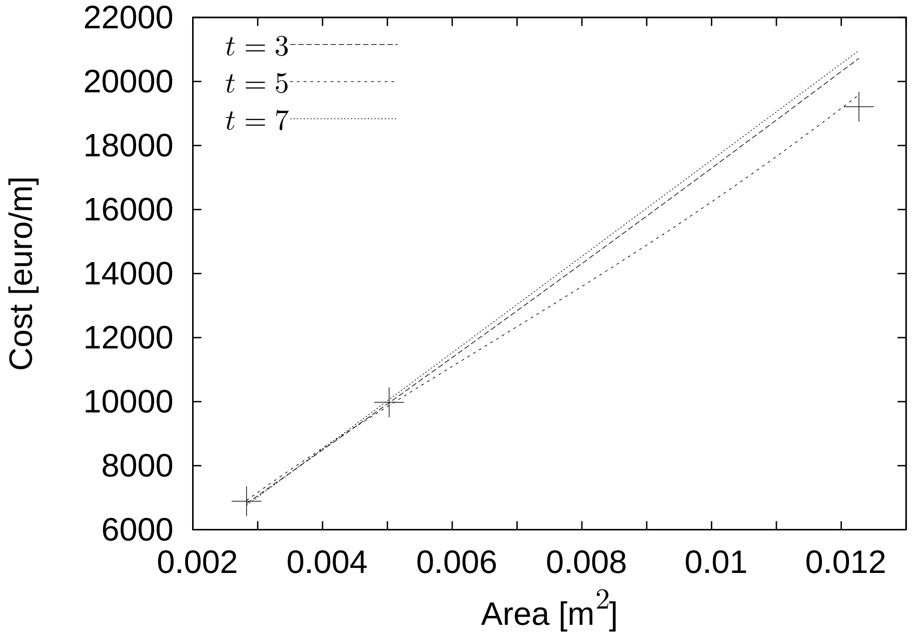
where t_e is the desired degree and $\beta(j,e)$ are the coefficients of the polynomial[2] approximating C_e.

We have experimented with several low-degree polynomials in order to find a satisfactory approximation and three values of t, namely $t = 3, 5, 7$, are depicted in Figure 7.1. In particular, the figure compares the polynomials with the discrete cost values, depicted as "+", for the instance `foss_poly_0` (see Section 7.5.1) and by taking into account the three

[2]Note that the least-square minimization is by itself a non-convex NLP that we solve to a local optimum by the open-source NLP solver `Ipopt` ([123]), in turn used as NLP solver all over the chapter (see Section 7.5.2).

smallest discrete diameter sizes, i.e., the ones that are mostly used in high quality solutions. Note that for each pipe of this instance, the set of diameters is the same, thus we used the same continuous cost function for each pipe. The polynomial that best fits our purposes especially for these important diameters is the one of degree 5. Note that, we do not insure that the polynomial is increasing nor convex, and actually we do not even assume this for the data, though for the data sets that we experimented with the discrete data are increasing.

Figure 7.1: Three polynomials of different degree approximating the cost function for instance `foss_poly_0`, see Section 7.5.1.



We will come back to the choice of discrete vs continuous objective function in Section 7.4.2 and we will report some computational experiments in Section 7.5.2.

Before ending the section we note that one drawback to using a low-degree polynomial (for each pipe) to fit the discrete costs is that this would attain the correct value of the objective function for each integer solution only if there is a low-degree polynomial that has a relative error equal to 0. As this is unlikely, we may have to make a compromise, in modeling the objective function, between modeling accuracy and numerical behavior.

This difficulty can be overcome in an alternative manner. We can instead define a continuous objective function so as to fit the discrete values $\mathcal{C}(e, r)$ using a cubic spline for each pipe e. Each piece of a cubic spline is a degree-three polynomial that passes between a pair of consecutive discrete points $(\mathcal{D}(e, r-1)$,$\mathcal{C}(e, r-1))$ and $(\mathcal{D}(e, r)$, $\mathcal{C}(e, r))$ $(e \in E$, $r = 2, \ldots, r_e)$. The use of cubic splines guarantees that, once an integer solution is found, its objective value is correct.

This piecewise definition of the function can be easily accommodated using a modeling language like AMPL (which has a natural syntax for defining piecewise functions). However, the NLP solvers, and in particular `Ipopt` (see Section 7.5.2), seem to more easily manage a polynomial with high degree, as compared to $r_e - 1$ different polynomials pieces of degree 3, thus the experiments in Section 7.5 use the single polynomial objective function with an algorithmic correction for taking into account the original discrete one (see Section 7.5.2). We do note, however, that we believe that the spline approach has considerable potential, but more work would be needed on the side of NLP solvers to realize a computational benefit.

7.3.1 Smoothing the nondifferentiability

The main remaining modeling difficulty is to deal algorithmically with the absolute value term in the head loss constraints. This term is nondifferentiable (at 0) but not badly so. One possibility is to ignore the nondifferentiability issue, and just use a solver that will handle it in its own way. This has the advantage of straightforward implementation from AMPL and access to many NLP solvers (e.g., via NEOS ([96])). Because our ultimately goal is, however, to employ available MINLP solvers, we tested such a straightforward approach by using the MINLP solver `Bonmin` and its default NLP solver `Ipopt` (see Section 7.5.2). The result is rather disappointing: `Ipopt` is unable to deal with such nondifferentiable function and it aborts the run immediately.

Thus, to accommodate the NLP solver, we had to smooth the nondifferentiability, and in order to do that effectively, our main goal is not to provide a fully accurate approximation near 0 because it is well known that Hazen-Williams equation is in itself a poor approximation of the real pressure loss for small values of the flow. Instead, we smooth away the mild nondifferentiability by defining the head loss equation in a piecewise manner, in such a manner as to have accurate evaluations of the function. We insure the smoothness by matching function values as well as first and second derivative at the breakpoints.

More precisely, let $f(x) = x^p$ ($p = 1.852$) when x is nonnegative, and $f(x) = -f(-x)$ when x is negative (x is standing in for $Q(e)$). This function misbehaves at 0 (the second derivative does not exist there). Choose a small positive δ, and replace f with a function g on $[-\delta, +\delta]$. Outside of the interval, we leave f alone. We will choose g to be of the following form: $g(x) = ax + bx^3 + cx^5$. In this way, we can choose a, b, c (uniquely) so that f and g agree in value, derivative and second derivative, at $x = |\delta|$. So we end up with a smooth-enough anti-symmetric function. It agrees in value with f at 0 and outside $[-\delta, +\delta]$. It agrees with f in the first two derivatives outside of $[-\delta, +\delta]$.

Formally, it is easy to prove that:

Proposition 7.1.. *The unique polynomial $g(x) = ax + bx^3 + cx^5$ having $f(x) = g(x)$, $f'(x) = g'(x)$ and $f''(x) = g''(x)$ at $x = |\delta|$ is:*

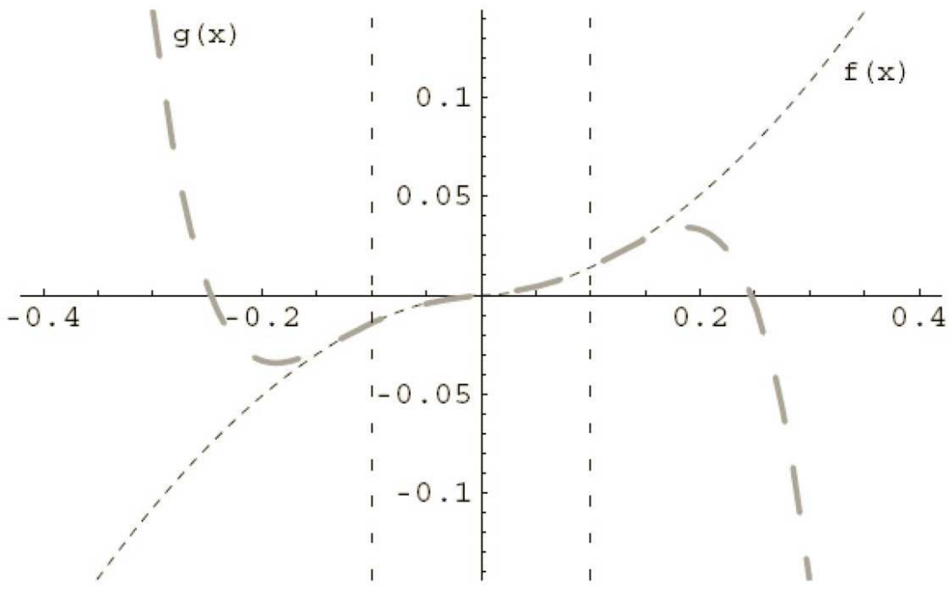
$$g(x) = \left(\frac{3\delta^{p-5}}{8} + \frac{1}{8}(p-1)p\delta^{p-5} - \frac{3}{8}p\delta^{p-5}\right)x^5 + \left(-\frac{5\delta^{p-3}}{4} - \frac{1}{4}(p-1)p\delta^{p-3} + \frac{5}{4}p\delta^{p-3}\right)x^3 + \left(\frac{15\delta^{p-1}}{8} + \frac{1}{8}(p-1)p\delta^{p-1} - \frac{7}{8}p\delta^{p-1}\right)x\ .$$

Proof. Via simple calculation one simply has to equate: (i) $g(\delta) = a\delta + b\delta^3 + c\delta^5 = \delta^p = f(\delta)$, (ii) $g'(\delta) = a + 3b\delta^2 + 5c\delta^4 = p\delta^{p-1} = f'(\delta)$, and (iii) $g''(\delta) = 6b\delta + 20c\delta^3 = p(p-1)\delta^{p-2} = f''(\delta)$. This is now a square linear system in the a, b, c variables. We solve it (symbolically), using *Mathematica* (see [90]).

Finally, we just observe that f and g are anti-symmetric, so we have the same a, b, c for $x = -\delta$. □

Figure 7.2, drawn for $\delta = 0.1$, shows that g provides a good approximation of f. Indeed the quintic curve fits very well on $(-\delta, +\delta)$, and of course it matches up to second order with the true function f at $\pm\delta$. This is all no surprise because we are operating in a small interval of 0, and the function that we approximate is not pathological. The NLP solvers that we have tested appear to respond well to this technique, as does our MINLP solver itself, `Bonmin`.

Figure 7.2: Smoothing f near $x = 0$.



Piecewise constraints can be modeled in AMPL (see Section 18.3 of [55]), so we have the advantage of being able to use a variety of NLP solvers, as well as a path to using `Bonmin` and `MINLP_BB`, both of which are interfaced with AMPL. Our experience is that the inaccuracy in using this smoothed function is minimal compared to the other inaccuracies (e.g., numerical and modeling inaccuracies).

7.4 Models and algorithms

In this section we discuss how to turn our preliminary continuous NLP model into a MINLP that behaves well computationally. For this purpose, we analyze some relevant literature and we then discuss the discrete component of the problem.

7.4.1 Literature review

Optimal design of a WDN has already received considerable attention. [9] linearize and use an MILP approach. [112] and [41] work within an accurate mathematical model, but they use meta-heuristic approaches for the optimization, and they work with the constraints by numerical simulation. [58] employ a so called “split-pipe model” in which each pipe e is split into r_e stretches of unknown length, where r_e is the number of possible choices of the diameter of pipe e, and variables model the lengths of the stretches. It is not difficult to see that models of this type have the disadvantage of allowing solutions with many changes in the diameter along the length of a pipe. Furthermore, there can be additional pressure loss at junctions of the stretches (so called “minor head losses”) which could become significant if there are many different stretches along a pipe; such losses are ignored by all optimization models that we know of. Using this type of split-pipe model, [58] employ a meta-heuristic approach for the optimization, working with the constraints by numerical simulation. [47] also work with a split-pipe model, but they use NLP methods for calculating a solution. [116] also work with a split-pipe model, and they successfully employ global optimization methods. Of course, global optimization methods may become impractical for very large scale instances. [79] and [131] also employ an NLP approach, but they use an approximation of the split-pipe methodology (using just two pipe sections). Because the split-pipe model is a relaxation of ours (we only allow a single choice of diameter along the length of a pipe), results using such a model are not directly comparable to ours.

In the rest of the chapter, we develop an MINLP approach and compare it to the MILP approach of [9]. The MILP approach has the advantage of correctly modeling the choices of discrete diameters with binary indicator variables $X(e, r)$ representing the assignment of diameter $\mathcal{D}(e, r)$ to arc e. In this way we can also easily incorporate costs for the chosen diameters. There is still the non-linearity of the flow terms in the head loss constraints. Piecewise-linear approximation of these non-linear constraints is the standard MILP approach here. Unfortunately, the resulting MILPs are typically very difficult to solve. The difficulty of the MILP models is related to the fact that once the diameters have been fixed, the objective function is set, and a feasibility problem associated with the piecewise-linear approximation must be solved, without any guidance from the objective function. It turns out that linear-programming tools in such a context are not effective at all. Good feasible solutions to the models are not always obtainable for even networks of moderate size. Often one is lead to using very coarse piecewise-linear approximations to get some sort of solution, but these tend to not be accurate enough to be considered truly feasible. Indeed, especially with few linearization points, the MILP may (i) generate flows that are not compatible with the selected diameters because the relation between these variables is only approximated (so the flows computed with the real functions may well be infeasible), and (ii) cut off some feasible (and potentially optimal) solutions. Section 7.5 includes some of these rather negative computational results obtained with the MILP approach.

7.4.2 Discretizing the diameters

We need an effective method for imposing the restriction that the diameter of each pipe $e \in E$ belongs to the discrete set of elements:

$$d_{min}(e) := \mathcal{D}(e, 1) < \mathcal{D}(e, 2) < \cdots < \mathcal{D}(e, r_e) =: d_{max}(e) \ .$$

It would be natural and simple to handle this mostly at the level of the MINLP solver, just pass these discrete values to the MINLP solver `Bonmin` via the modeling language (AMPL), and let the MINLP solver construct a two-way branch for a continuous diameter that is strictly between an adjacent pair of discrete choices. Though we could make the necessary changes to the solver `Bonmin`, there does not appear to be a clean way for AMPL to pass such information to the solver. Of course this could be handled in an ad hoc manner, via an auxiliary file, but we prefer to do things in a manner that can be easily and naturally applied to other MINLP solver.

So, for the present, we simply define additional binary variables $X(e,i)$, where $X(e,i) = 1$ indicates that diameter $\mathcal{D}(e,r)$ is selected for pipe e ($r = 1,\ldots,r_e$, $e \in E$). Then, we use the "SOS type-1" branching (see, [13]) that is available in `Bonmin v. trunk`. As is standard, we use AMPL suffixes to pass along the SOS information needed by the solver: .sosno ("SOS number") is used to group variables into separate SOS constraints and .ref ("reference value") is used to indicate the value symbolized by a variable. In this way, for $e \in E$, in AMPL we naturally set:

$$X(e,r).sosno := e \ , \ \text{for } r = 1,\ldots,r_e \ ,$$

and

$$X(e,r).ref := \mathcal{D}(e,r) \ , \ \text{for } r = 1,\ldots,r_e \ .$$

We note that with the introduction of these binary variables, we could use them in the objective function and eliminate the need for the fitted objective function introduced in Section 7.2. However, to do so would implicitly define a piecewise-linear cost function for each pipe, and because of our reliance on NLP solvers that prefer smooth functions, we stay with our method of handling the objective. Also, eventually we hope to eliminate the need to introduce these binary variables, in which case our approach for the objective function would still be required. In any case, a detailed computational comparison between the fitted objective function and the discrete one is given at the end of Section 7.5.2.

Finally, we remark that in the preliminary report on our work ([30]), we described a different method for handling the discrete nature of the diameters. At that time, `Bonmin` was not yet able to handle SOS constraints, so we attempted to approximate the behavior of SOS branching via a different definition of binary variables and a judicious setting of branching priorities.

7.4.3 Parameterizing by area rather than diameter

We can use variables:

$$A(e) = \text{cross-sectional area of pipe } e \ (\forall \ e \in E),$$

rather than the diameter variables $D(e)$ ($e \in E$). This allows us to eliminate the non-linearities and non-convexities of the flow bounds which then become:

$$-v_{max}(e)A(e) \leq Q(e) \leq v_{max}(e)A(e) \quad (\forall \ e \in E).$$

The other constraints remain substantially similar. The simple bounds become:

$$\tfrac{\pi}{4}d_{min}^2(e) \leq A(e) \leq \tfrac{\pi}{4}d_{max}^2(e) \quad (\forall \ e \in E),$$

and the head loss across links constraints are:

$$H(i) - H(j) = \operatorname{sgn}(Q(e))|Q(e)|^{1.852} \cdot 10.7 \cdot len(e) \cdot k(e)^{-1.852} \left(\tfrac{\pi}{4}\right)^{2.435} / A(e)^{2.435}$$
$$(\forall \ e = (i,j) \in E).$$

Although the head loss equations remain non-linear, note that the effect of the use of areas instead of diameters is a perceptible reduction of the exponent of the design variables. Namely, we have $A(e)^{2.435}$ versus $D(e)^{4.87}$. Ultimately, however, the decision between the area and the diameter parametrization is, as usual, computational. We compared the two approaches with computational experiments and the diameter approach returned the same solution as the area one for 3 instances over 9, a worse solution in other 3 instances and a better one in the remaining 3 instances. Overall, the average of the percentage deviation[3] of the best MINLP solutions computed for the diameter approach with respect to those obtained with using the area is 3.91. In addition, the average computing time to find the best solution for the 3 instances reporting the same solution (namely `hanoi`, `blacksburg` and `foss_poly_0`, described in Section 7.5.1) is 1,132 seconds for the area formulation, and 2,242 seconds for the diameter one. Although these results do not show a strict domination, we do prefer the area parametrization that appears ultimately more stable and better from a mathematical point of view. Thus, in Section 7.5.2 we report the results using the area parametrization.

7.5 Computational experience

In this section we give detailed computational results on instances from both the literature and real-world applications. These results are compared to previously reported solutions (by sometimes discussing their accuracy). We also discuss in detail the use of an open-source MINLP software, namely `Bonmin`. Finally, we also report some (unsatisfactory) computational results on the MILP models obtained with the technique of [9], and we highlight why an MINLP approach is in this case far superior to the MILP counterpart.

7.5.1 Instances

Our data comprises 9 instances that capture different aspects of real-world networks. In particular, these instances vary in size, type, number and diameter of the pipes that can be installed. Moreover, some special requirements to be discussed below are sometimes present.

The main characteristics of the instances are reported in Table 7.1.

For each instance, Table 7.1 reports the name and the numbers of junctions (including the reservoirs), reservoirs, pipes and diameter choices. Moreover, the column labeled "duplicates" indicates the number of pipes whose diameter is fixed but which can possibly be duplicated by installing a new pipe (whose diameter must be determined) in parallel. Finally, the last column indicates which currency is used to express the unit cost of the pipes, namely, US Dollar ($), Italian Lira (£) and Euro (€).

Instances `shamir`, `hanoi`, `blacksburg` and `New York` are taken from the literature, while the others are real-world instances of Italian water networks[4].

For the instances from the literature, the only one that requires some preprocessing of the data in order to fit into our definitions is `New York` which will be separately discussed below. However, the data for instance `blacksburg` available from [116] was incomplete, and the final version of the instance that we used and make available is certainly (slightly) different from the original one.

[3]The percentage deviation of algorithm A with respect to algorithm B is computed as $100 \times (value[A] - value[B])/value[B]$.

[4]All instances are available at `www.or.deis.unibo.it/research_pages/ORinstances/ORinstances.htm`.

Table 7.1: Water Networks.

	number of ...					unit
name	junctions	reservoirs	pipes	duplicates	diameters	cost
shamir	7	1	8	–	14	$
hanoi	32	1	34	–	6	$
blacksburg	31	1	35	–	11	$
New York	20	1	21	21	12	$
foss_poly_0	37	1	58	–	7	£
foss_iron	37	1	58	–	13	€
foss_poly_1	37	1	58	–	22	€
pescara	71	3	99	–	13	€
modena	272	4	317	–	13	€

For the real-world instances, the three instances "foss_X" refer to a single neighborhood of Bologna, called Fossolo. Instance foss_poly_0 consists of the original data provided to us and the pipe material for that instance is polyethylene. Instance foss_iron is for the same network, but with almost twice as many choices of pipe diameters and with the material being cast iron. For instance foss_poly_1 the material for the pipes is again polyethylene but there are more choices than foss_poly_0 for the pipe diameters.

The cost data for foss_poly_0 is out of date, and so the solution values cannot be directly compared to those of foss_poly_1 and foss_iron, which, in turn, can be reasonably compared. The value of the solution reported in Section 7.5.2 for foss_poly_1 is much lower than for foss_iron. At first this seems surprising, but it is because polyethylene is much cheaper than cast-iron.

Finally, pescara and modena are reduced versions of the water distribution networks of two medium-size Italian cities. The pipe material is cast iron and both costs and diameters are up-to-date values in the Italian market.

The famous New York instance

The New York instance was first introduced by [114]. The problem we need to solve for this instance is quite different from the original one. Given an existing network, the objective is to "renovate" it by considering the increase of the water demand due to the population growth. The existing network is no longer adequate for the increased demand, resulting in pressure violations at several junctions. Thus, the network must be modified by duplicating some of the pipes, i.e., putting new pipes in parallel with the existing ones, at a minimum cost.

The decisions one has to take are:

1. select the pipes that need to be duplicated;
2. for each of these pipes, choose a diameter within the available diameter set.

In other words, with respect to our model, one has to add the null value to the diameter set: if such a null value is selected, it corresponds to the reverse of the decision 1 above, i.e., the pipe is not duplicated. However, such an explicit addition of the null diameter requires relevant modifications (consider the head loss equations), and an overall deterioration of our model.

Thus, we decided to handle such a case by an alternative method, along a line proposed by [114]. Note that this approach was not presented and formally stated in [114], but it can be read from the code reported in that paper. For the sake of clarity and completeness, we report it explicitly here.

The idea is to transform the problem, that includes the two decisions above, into our original problem, thus avoiding the first decision. We can easily do it introducing the *equivalent pipe* concept: We treat implicitly the two parallel pipes by means of a unique equivalent pipe that reproduces the same behavior at the extreme junctions of the pipe within the network. For each diameter of the duplicated pipe (including the null one) there is a discrete equivalent diameter associated with the pair existing/duplicated pipes.

We can prove the following simple result:

Theorem 7.2.. *For each pipe $e \in E$ the new diameters and costs are, respectively:*

$$\begin{aligned}\mathcal{D}_{new}(e,r) &= \left(D_{fix}(e)^{\frac{4.87}{1.852}} + \mathcal{D}(e,r)^{\frac{4.87}{1.852}}\right)^{\frac{1.852}{4.87}}\\ \mathcal{C}_{new}(e,r) &= \mathcal{C}(e,r)\ ,\end{aligned}$$

with $r = 0, 1, \ldots, r_e$ and where $D_{fix}(e)$ is the diameter of the existing pipe and $\mathcal{D}(e,0) = \mathcal{C}(e,0) = 0$.

Proof. Formally, for each existing pipe $e \in E$, we add two pipes e' and e'' corresponding to the duplicated and equivalent pipes, respectively. First, note that the flow through the existing and duplicated pipes must follow the same direction because they have the same start and end junctions and, consequently, the same hydraulic head which determines the flow direction. Thus, $Q(e)$, $Q(e')$ and $Q(e'')$ agree in sign and denote the flows over the corresponding pipes. In order to impose the above described equivalence we must solve the following system of equations:

$$\begin{aligned}&Q(e) + Q(e') = Q(e'')\\ &H(i) - 10.7 \cdot Q(e)^{1.852} \cdot k(e)^{-1.852} \cdot D(e)^{-4.87} \cdot len(e) - H(j) = 0\\ &H(i) - 10.7 \cdot Q(e')^{1.852} \cdot k(e')^{-1.852} \cdot D(e')^{-4.87} \cdot len(e') - H(j) = 0\\ &H(i) - 10.7 \cdot Q(e'')^{1.852} \cdot k(e'')^{-1.852} \cdot D(e'')^{-4.87} \cdot len(e'') - H(j) = 0\ .\end{aligned}$$

As required, these equations guarantee that, substituting the two parallel pipes with the equivalent one, we obtain the same flow and the same head loss at the start and end junctions.

The system above can be easily simplified by substituting out the flows:

$$\begin{aligned}\left(\frac{H(i)-H(j)}{10.7 \cdot len(e)}\right)^{\frac{1}{1.852}} \cdot k(e) \cdot D(e)^{\frac{4.87}{1.852}} + \left(\frac{H(i)-H(j)}{10.7 \cdot len(e')}\right)^{\frac{1}{1.852}} \cdot k(e') \cdot D(e')^{\frac{4.87}{1.852}}\\ = \left(\frac{H(i)-H(j)}{10.7 \cdot len(e'')}\right)^{\frac{1}{1.852}} \cdot k(e'') \cdot D(e'')^{\frac{4.87}{1.852}}\ .\end{aligned}$$

Because $len(e) = len(e') = len(e'')$ and, in this instance, $k(e) = k(e') = k(e'')$, it is easy to see that:

$$D(e'') = \left(D(e)^{\frac{4.87}{1.852}} + D(e')^{\frac{4.87}{1.852}}\right)^{\frac{1.852}{4.87}}\ ,$$

which proves the result. □

7.5.2 MINLP results

We have tested our approach using the open-source MINLP solver `Bonmin` (see [20, 24]) which is distributed on COIN-OR. In the following we describe the basic features of the solver and we report the computational results on the instances described in Section 7.5.1.

`Bonmin` B&B algorithm description

`Bonmin` (Basic Open-source Non-linear Mixed INteger programming) is an open-source code for solving MINLP problems of the form:

$$\begin{aligned}
&\min\ f(x)\\
&g^L \le g(x) \le g^U\\
&x^L \le x \le x^U\\
&x \in \mathbb{R}^n\\
&x_i \in \mathbb{Z}\ , \quad \forall\ i \in I\ ,
\end{aligned}$$

where the functions $f : \{x \in \mathbb{R}^n : x^L \le x \le x^U\} \to \mathbb{R}$ and $g : \{x \in \mathbb{R}^n : x^L \le x \le x^U\} \to \mathbb{R}^m$ are assumed to be twice continuously differentiable, and $I \subseteq \{1, \ldots, n\}$.

There are several algorithms implemented within `Bonmin`:

B-BB, a simple Branch-and-Bound algorithm based on solving a continuous non-linear program at each node of the search tree and branching on variables;

B-OA, an outer-approximation based decomposition algorithm;

B-QG, an outer-approximation based Branch-and-Bound algorithm;

B-Hyb, a hybrid outer-approximation/non-linear programming based Branch-and-Cut algorithm.

The different methods that `Bonmin` implements are exact algorithms when the functions f and g are convex but are only heuristics when this is not the case. For an MINLP having a non-convex relaxation (like the WDN problem), the B-BB algorithm should be used because the outer-approximation constraints are not necessarily valid inequalities for the problem. Although even B-BB is only a heuristic for such a non-convex problem (the NLP problems at each node are not solved to global optimality), `Bonmin` includes several options tailored to improve the quality of the solutions it provides. First, in the context of non-convex problems, `Ipopt` (the `Bonmin` default NLP solver) may end up in different local optima when started from different starting points. The two options `num_resolve_at_root` and `num_resolve_at_node` allow for solving the root node or each node of the tree, respectively, with a user-specified number of different randomly-chosen starting points, saving the best solution found. Note that the function to generate a random starting point is very naïve: it chooses a random point (uniformly) between the bounds provided for the variable.

Secondly, because the solution given by `Ipopt` does not truly give a lower bound, the user can adjust the fathoming rule to continue branching even if the solution value to the current node is worse than the best-known solution. This is achieved by setting `allowable_gap`, `allowable_fraction_gap` and `cutoff_decr` to negative values.

In the next section, we will describe how we used the options of `Bonmin` designed for non-convex problems and some modifications we implemented in `Bonmin` specifically tailored for our WDN problem.

Improving `Bonmin` for WDN problems

As discussed in the previous section, `Bonmin` was originally developed for finding globally-optimal solutions to MINLPs having convex relaxations. However, some accommodations were made to handle non-convex instances as well, already in the released version `Bonmin v. 0.1`. In fact, these accommodations were developed and tested in the context of the present study. Additionally, we made and tested further modifications, to better handle non-convexities. We implemented these modifications starting from a copy of `Bonmin v. trunk`. The first modification proposed is available in the most recent stable versions of `Bonmin`. Eventually, the second modification may be adopted in a future release.

In particular, two main issues came up:

I.1 *Properly evaluating the objective value of integer feasible solutions.*
In Section 7.2 we have introduced a new objective function so as to approximate the correct (discrete) one. During preliminary computational experiments, we noted that such an approximation sometimes has the effect of rejecting integer feasible solutions having the approximated objective value worse than the incumbent but with a better value with respect to the correct objective function. Such a behavior has been corrected by allowing `Bonmin v. trunk` to work with *two objective functions*: the first one v_{fit} (i.e., the smooth continuous approximation) is used to guide the search, while the second one v_{disc} (i.e., the correct *discrete* objective) is used to evaluate integer feasible solutions, so as to avoid fathoming improving leaves. So, each time a new integer feasible solution, say x^{new}, is found, the value of the discrete objective function is computed. In this way, we work with two incumbent solutions, say x^{inc} and $\overline{x}^{inc}$ for v_{fit} and v_{disc}, respectively. For every feasible solution, we do update separately each of the incumbents, if needed. More precisely, if $v_{fit}(x^{new}) \geq v_{fit}(x^{inc})$ using a single objective function would have led us to discard the solution x^{new}. In fact, we do test $v_{disc}(x^{new})$ with respect to $v_{disc}(\overline{x}^{inc})$ and in case $v_{disc}(x^{new}) < v_{disc}(\overline{x}^{inc})$ we do update the discrete incumbent. In any case, v_{fit} is used as primary objective function for pruning fractional solutions at the nodes (see point I.2 below).

I.2 *Heuristically reducing the size of the search space.* The released version `Bonmin v. 0.1` statically defines two parameters: The parameter `cutoff` is the value of a known feasible solution (i.e., the incumbent), while `cutoff_decr` is the value by which every new feasible solution should be improved with respect to the current incumbent (i.e., the `cutoff` above). On non-convex problems, `cutoff_decr` is selected to be *negative* so as to act in a conservative manner with nodes whose continuous solution is not-too-much-worse than the current incumbent. However, we found out that such a static definition of `cutoff_decr` does not fit our purposes because it is hard to define a unique value for all instances. After preliminary computational testing, we implemented in `Bonmin v. trunk` the following policy: the root node continuous value is computed for 50 different starting points and `cutoff_decr` is set as:

$$\texttt{cutoff\_decr} := -V \cdot f(\sigma) \ , \tag{7.1}$$

where V is the average of the 50 root node continuous values, $\sigma \in [0,1]$ is the coefficient of variation (standard deviation divided by the mean) of those values, and

$$f(\sigma) := \begin{cases} .02 \ , & \text{if } \sigma < .1 \ ; \\ .05 \ , & \text{if } \sigma \geq .1 \ . \end{cases}$$

In other words, the parameter is set taking into account how much different the solutions at the root node are, so as to be more careful in fathoming a node when such a difference is large. The characteristics of the instances with respect to the 50 continuous solutions computed at the root node using different random starting points are given in Table 7.2. More precisely, Table 7.2 reports for each instance the mean (mean), percentage deviation of the first solution found (% dev first), percentage deviation of the minimum (% dev min) and the percentage deviation of the maximum (% dev max) value of the continuous solution at the root note over the 50 samples. The table then reports the standard deviation (std dev), the coefficient of variation (coeff var) and finally the number of failures of `Ipopt` (# fail) and the number of times the continuous problems turned out to be infeasible (# inf). Table 7.2 demonstrates that the way we have

Table 7.2: Characteristics of the 50 continuous solutions at the root node.

	mean	% dev. first	% dev. min	% dev. max	std dev	coeff var	# fail	# inf
`shamir`	413,507.00	-1.502	-2.922	62.827	37,735.10	0.0912563	0	0
`hanoi`	6,112,600.00	-0.681	-2.271	2.114	88,473.50	0.0144740	0	0
`blacksburg`	114,534.00	-0.975	-0.975	7.084	1,659.97	0.0144932	0	0
`New York`	83,480,900.00	-53.278	-53.278	34.331	12,024,900.00	0.1440440	0	0
`foss_poly_0`	78,080,900.00	1.279	-12.094	51.136	11,096,800.00	0.1421190	0	0
`foss_iron`	181,977.00	-0.284	-0.757	4.207	9,081.52	0.0169336	0	0
`foss_poly_1`	33,076.40	-19.091	-19.459	55.683	5,470.00	0.1653750	0	0
`pescara`	1,846,930.00	-1.338	-1.338	16.622	66,672.00	0.0360989	0	0
`modena`	2,567,680.00	0.008	-0.103	0.421	1,920.57	0.0007480	0	0

modeled the problem has a stable behavior, in the sense that the continuous solutions never have numerical difficulties nor turn out to be infeasible. On the other hand, the solution value depends a lot on the starting point, and the most unstable instances are `New York`, `foss_poly_0` and `foss_poly_1`.

MINLP solutions

The results obtained using `Bonmin v. trunk` are reported in Tables 7.3 and 7.4, running the code with a time limit of 7,200 CPU seconds on a single processor of an Intel Core2 CPU 6600, 2.40 GHz, 1.94 GB of RAM under Linux. In particular, Table 7.3 reports the best solution values computed for the instances in the testbed. The value of the best solution with the fitted objective function (denoted as $v_{fit}(x^{best})$) is compared with the value of the best solution found with respect to the true objective function (denoted as $v_{disc}(\overline{x}^{best})$). In addition, we report the percentage deviation of the value of the solution $\overline{x}^{best}$ once mapped on the fitted objective function (denoted as % dev $v_{fit}(\overline{x}^{best})$). In particular, we marked (with a "✓") in the last column the three instances for which values $v_{fit}(x^{best})$ and $v_{fit}(\overline{x}^{best})$ are different, i.e., the instances in which the use of both objective functions simultaneously had a very positive effect. Note that a positive value for the percentage deviation indicates an improvement: the (true) best solution would not have been found without modification I.1 because its fitted value was in fact worse than the fitted value of the incumbent used by the algorithm.

Table 7.4 reports additional results on the same instances and with the same tuning of the code. In particular, besides the best MINLP solution value with respect to the true objective function ($v_{disc}(\overline{x}^{best})$) (same column of Table 7.3), we report the CPU time in seconds to

Table 7.3: Computational results for the MINLP model (part 1). Time limit 7200 seconds.

	$v_{fit}(x^{best})$	$v_{disc}(\overline{x}^{best})$	% dev. $v_{fit}(\overline{x}^{best})$	
`shamir`	423,696.31	419,000.00	0.000	
`hanoi`	6,109,620.78	6,109,620.90	0.000	
`blacksburg`	118,251.06	118,251.09	0.000	
`New York`	39,570,174.42	39,307,799.72	0.541	✓
`foss_poly_0`	70,842,869.58	70,680,507.90	0.000	
`foss_iron`	181,865.00	178,494.14	0.024	✓
`foss_poly_1`	29,062.82	29,202.99	0.000	
`pescara`	1,883,480.00	1,837,440.40	0.187	✓
`modena`	2,620,189.45	2,580,379.53	0.000	

find such a solution (time) and percentage deviation of the best MINLP solution value after 1,200 CPU seconds (denoted as % dev $v_{disc}(\overline{x}^{first})$, recall that the overall time limit is 7,200 CPU seconds). Finally, the last two columns report the number of updates of the incumbent solution value for the fitted (# fit) and true (# true) objective function, respectively. When such numbers are different for the same instance, it means that using simultaneously two objective functions had an effect, i.e., it changed the explored tree. Such instances are a superset of the three marked in Table 7.3 with a "✓" in the last column. Precisely, the

Table 7.4: Computational results for the MINLP model (part 2). Time limit 7200 seconds.

	$v_{disc}(\overline{x}^{best})$	time	% dev. $v_{disc}(\overline{x}^{first})$	# fit	# true
`shamir`	419,000.00	1	0.000	2	2
`hanoi`	6,109,620.90	357	0.000	8	8
`blacksburg`	118,251.09	1,540	0.178	6	6
`New York`	39,307,799.72	3	0.000	5	6
`foss_poly_0`	70,680,507.90	1,500	0.058	6	8
`foss_iron`	178,494.14	3,070	0.350	4	5
`foss_poly_1`	29,202.99	6,772	0.274	6	5
`pescara`	1,837,440.40	6,701	0.447	7	21
`modena`	2,580,379.53	964	0.000	2	2

effect of modification I.1 above is crucial for the three instances `New York`, `foss_iron` and `pescara` in which the final solution of the fitted objective function is not the best one with respect to the discrete objective function. Moreover, Table 7.4 demonstrates that besides the three above instances, the use of the two objective functions is also effective in the two other `fossolo` instances where during the search some solutions with good value of the discrete objective function are kept. These solutions are not the best ones at the end of the 2 hours time limit, but clearly they could have been with a different time limit.

The effect of modification I.2, instead, is an improvement on instances `foss_poly_0` and `foss_poly_1`: specifically, the solution of the former strongly improves from 71,595,394.14 to 70,680,507.90 while the improvement of the latter is smaller (from 29,226.71 to 29,202.99).

Note that the solutions obtained within 20 minutes of CPU time (see Table 7.4) are also very good and show how the MINLP search is quite effective also for short computing times.

The overall behavior of the code is dependent on the search options that can be selected among the `Bonmin v. trunk` arsenal. In particular, the reported results are all obtained with `tree_search_strategy = dive` and `node_comparison = best-bound` which turned out to give the most stable and effective version. On the other hand, slightly better results on single instances can be obtained with different parameters and in particular using `node_comparison = dynamic`.

A natural computational question is the impact of using the fitted objective function as compared to using only the discrete one. We performed an additional set of experiments with the original discrete function and the results are reported in Table 7.5. For the discrete

Table 7.5: Computational results for the MINLP model comparing the fitted and the discrete objective functions. Time limit 7200 seconds.

	$v_{disc}(\overline{x}^{best})$	time	% dev. "discrete"	time
`shamir`	419,000.00	1	0.00	3
`hanoi`	6,109,620.90	357	0.00	1,059
`blacksburg`	118,251.09	1,540	0.00	1,384
`New York`	39,307,799.72	3	0.00	217
`foss_poly_0`	70,680,507.90	1,500	0.00	2,502
`foss_iron`	178,494.14	3,070	0.00	5,584
`foss_poly_1`	29,202.99	6,772	-0.35	681
`pescara`	1,837,440.40	6,701	1.17	576
`modena`	2,580,379.53	964	1.69	106

case, we report in Table 7.5 the percentage deviation of the best solution (denoted as % dev "discrete") with respect to our best results $v_{disc}(\overline{x}^{best})$ and the corresponding computing time to achieve the best solution. The table shows that on 6 of the 9 instances the final solution using the discrete diameters is the same, but the average computing time is 1,033.0 seconds instead of 680.2, i.e., it requires a bit more time on average. For 2 of the 3 remaining cases, a worse solution value is found. Notably, these worse solutions are for our largest instances. Finally, a slightly better solution is instead obtained in a shorter computing time for only one instance: `foss_poly_1`. In summary, the algorithm using the discrete objective function performs somewhat worse than the one with the fitted one and in particular this happens for the largest instances. This behavior is no surprise: in the discrete case the continuous solution obtained by relaxing integrality is always much smaller than the one for the fitted objective function because the piecewise-linear function can use non-consecutive discrete values in the convex combination. This results in a much larger search space to be explored because bad nodes might not be fathomed.

Finally, concerning the time limit of 7,200 CPU seconds, we note that it is reached in all reported tests with different settings and options, the only exception being the instance `shamir`. In such a case, once the solution of value 419,000 is found all active nodes have a continuous value larger by at least 2%, thus they are all fathomed.

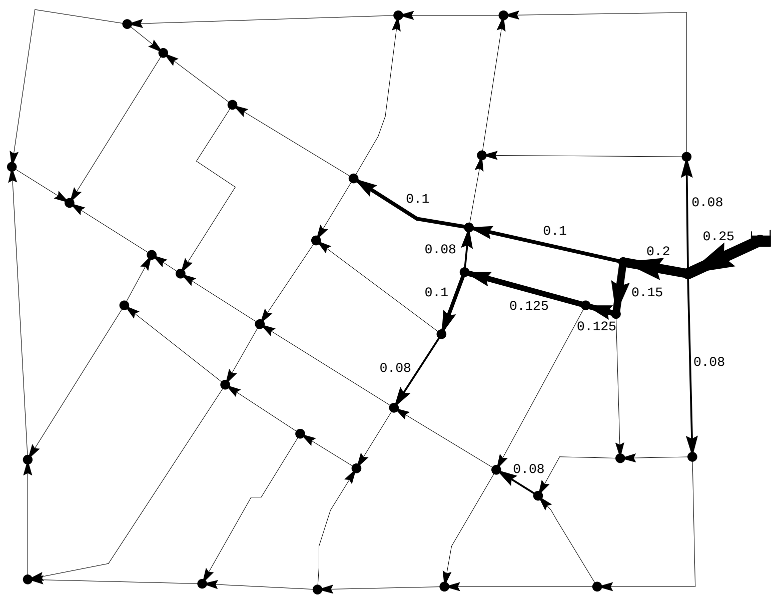
Practical use of the MINLP solutions

The analysis of the best MINLP solutions for the considered instances shows configurations in which the size of the selected diameters decreases from the reservoir toward the parts of the network farther away from the inlet point. This characteristic of the allocation of diameters to pipes plays in favor of a correct hydraulic operation of the network and has a beneficial effect on water quality, see, e.g., the discussion in [121]. This is depicted in Figure 7.3 where the size of each diameter is proportional to the thickness of the arc[5]. It is

[5]In Figure 7.3 diameters are expressed in meters, and the diameter is equal to 0.06 for the pipes without explicit number, i.e., the minimum diameter permissible for this data set.

easy to see that there are no pipes having large diameters isolated in the network. Such a characteristic, very appreciated by practitioners, does not normally occur when the design is done by a sort of "generate and test" approach in which configurations are produced within a naïve genetic-algorithms framework and then simulated for feasibility by using EPANET ([48])[6]. However, that is the approach commonly used in practice, and it requires considerable postprocessing for correcting configurations that are trivially non-optimal with pipes having diameter significantly different from the surrounding ones. In such a postprocessing, the arcs might be analyzed one-by-one in the attempt of reducing the size of their assigned diameters. We denote as *1-optimal* a solution in which the size of the diameter $\mathcal{D}(e,i)$ of a single pipe e cannot be reduced to the value $\mathcal{D}(e,i-1)$. Instead, the structure of the solutions obtained

Figure 7.3: Solution for Fossolo network, version `foss_iron`.



by our algorithm reflects such a general design criterion widely used by practitioners and represents an interesting indicator of the strength of the proposed approach. The solution depicted in Figure 7.3 is ready to be used in practice without any postprocessing, i.e., it is 1-optimal. In practice, it never happened in our large set of experiments that the algorithm returned as a best incumbent a non 1-optimal solution.

Literature comparison

The comparison with "existing numbers", i.e., with previous known solutions for the benchmark instances, is in general very difficult because of different parameter sets and coefficients used for the Hazen-Williams formula (see, e.g., the discussion in [112]). Despite such a difficulty, we report the following results.

[6]EPANET is free software distributed by the US Environmental Protection Agency.

1. For the small instance `shamir`, we find the previously best known (and probably optimal) solution.

2. On instance `blacksburg` we are not able to compare our results because: (i) the only results are those reported in [116] which are obtained with the split-pipe approach and (ii) part of the data was missing and, as mentioned in Section 7.5.1, the instance we use is different from the one used in that paper.

3. As stated in Section 7.2, the set of coefficients we decided to use for the Hazen-Williams equation is the one of [124]. In order to provide an informative comparison with other authors on the remaining instances, namely `hanoi` and `New York`, we ran our code by using each time the coefficient set proposed in the paper to be compared. In particular, in Table 7.6 we compare our MINLP approach with the approaches in the following papers:

   - [112]. In such a paper the authors analyze the results previously reported in the literature for the problem and define two different sets of coefficients corresponding essentially to the most "relaxed" version of the empirical formula for the Hazen-Williams equation and to the most "restrictive" one. We consider in Table 7.6 both versions, denoted as "SW99 rel." and "SW99 res.", respectively.
   - [42], denoted as "DSM96".
   - [41], denoted as "CS99".

   The MINLP solution obtained by our model using the above sets of coefficients is as usual denoted as $v_{disc}(\overline{x}^{best})$ in Table 7.6. An entry "—" in the table denotes that a particular instance has not been considered in a specific paper.

   Table 7.6 shows that our MINLP approach is able to find a solution at least as good as the other approaches in all but one of the cases. More precisely, it improves three times, it matches the same result twice and it is slightly worse compared to "SW99 res." on instance `New York`. This is a very satisfactory behavior which proves that the approach is not affected by the coefficient sets used for the Hazen-Williams equation.

   Concerning the running times, in [112], the reported results are obtained by several runs of 3 hours for each instance on a PC 486/DX2 50 computer. In [42], each run is of 50 minutes on a Sun Sparc 1 + Station with the operating system SunOS 4.1. Finally, in [41], the runs are of 2 hours on a Pentium PC at 166 MHz.
   It is hard to compare these results because of the different computing platforms but our algorithm is able to find the solutions for all sets of coefficients of `hanoi` and `New York` within 1 minute of CPU time, thus showing a very competitive behavior.

MILP results

The only details that are needed to implement the [9] MILP approach for our problem concern the piecewise-linear approximation of the Hazen-Williams equations[7]. First, note that for every $e = (i, j) \in E$, we have to separately consider the case in which the flow goes from i to j or vice versa. In other words, for approximating the two parts of the curve described by

[7]The first implementation of this method was presented in [8].

Table 7.6: MINLP results compared with literature results.

	[112]		[112]		[42]		[41]	
	SW99 rel.	$v_{disc}(\overline{x}^{best})$	SW99 res.	$v_{disc}(\overline{x}^{best})$	DSM96	$v_{disc}(\overline{x}^{best})$	CS99	$v_{disc}(\overline{x}^{best})$
`hanoi`	6.073 e+06	6.052 e+06	6.195 e+06	6.183 e+06	—	—	6.056 e+06	6.056 e+06
`New York`	37.13 e+06	36.68 e+06	40.42 e+06	40.47 e+06	38.8 e+06	38.8 e+06	—	—

the Hazen-Williams equation, we need two separate sets of weights which are then combined by writing a unique SOS constraint of type 2.

Second, for each of the two parts above, say from i to j, we have to decide how to sample the curve so as to approximate it. In particular, for a fixed diameter value, we plot the flow on the y-axis as a function of the head loss $H(i) - H(j)$, on the x-axis. Then, we compute an upper bound on the head loss value as:

$$\Delta H_{ij}(e) = \min \Big\{ \max \Big\{ \, (ph_{max}(i) + elev(i)) - (ph_{min}(j) + elev(j)) \, ,0 \Big\} \, , \\ \max_{r=1,\ldots,r_e} \left\{ \frac{10.7 \cdot len(e)}{k(e)^{1.852} \cdot \mathcal{D}(e,r)^{4.87}} \left(\frac{\pi}{4} \mathcal{D}(e,r)^2 v_{max}(e) \right)^{1.852} \right\} \Big\} \, .$$

The second term of the above equation can be simplified by recalling that $\mathcal{D}(e,1) < \mathcal{D}(e,2) < \cdots < \mathcal{D}(e,r_e)$, and the bound can be rewritten as:

$$\Delta H_{ij}(e) = \min \Big\{ \max \Big\{ \, (H_{max}(i) - H_{min}(j)), 0 \Big\} \, , \\ \frac{10.7 \cdot len(e)}{k(e)^{1.852} \cdot d_{min}(e)^{4.87}} \left(\frac{\pi}{4} d_{min}(e)^2 v_{max}(e) \right)^{1.852} \Big\} \, ,$$

where $H_{max}(i) := ph_{max}(i) + elev(i)$ and $H_{min}(j) := ph_{min}(j) + elev(j)$.

The obtained interval $[0, \Delta H_{ij}(e)]$ is then split in two parts $[0, \frac{1}{3}\Delta H_{ij}(e)]$ and $[\frac{1}{3}\Delta H_{ij}(e), \Delta H_{ij}(e)]$. Within such intervals we perform uniform sampling by using the same number of linearization points. This means, of course, that we have a better approximation in the first part of the interval which is sensible because in the second part the curve is more flat, thus easy to approximate well with few points.

Note that, the analogous upper bound computed in the reverse direction from j to i, $\Delta H_{ji}(e)$, only differs in the first term above. Moreover, both bounds $\Delta H_{ij}(e)$ and $\Delta H_{ji}(e)$ are constant with respect to the diameter, i.e., the maximum value of the head loss on the x-axis is the same for every curve obtained by fixing the diameter value. In other words, the x-axis values of the linearization points are common to each diameter, while the corresponding y-axis value changes.

The computational results, obtained by using such an MILP model and [71] as MILP solver, are disappointing — in fact, this was our motivation for trying an MINLP approach. The MILP problems are difficult to solve mainly because once the diameters have been settled, the objective function is constant, and the model reduces to a feasibility problem which approximates a pure NLP. It is not surprising, then, that finding a feasible solution of a system of non-linear equalities with a standard MILP technique is not a good idea. Note that, when the diameters/areas are fixed, the feasibility problem reduces to a system of $|E| + |N \backslash S|$ equations in $|E| + |N \backslash S|$ variables, plus, in our model, there are inequalities corresponding to the bounds on the variables which have to be satisfied.

Moreover, the MILP approach is heavily dependent on the accuracy of the approximation of the non-linear component. If such an approximation is highly accurate, i.e., the number of linearization points is large, no MILP feasible solution – a solution which satisfies the constraints of the MILP model – can be found by [71] within reasonable CPU times: from a few hours for the small networks up to days for the larger ones. In other words, the models are not useful because they cannot find even feasible solutions in reasonable computing time.

Otherwise, with a rough approximation, the situation becomes even more complicated. Let us consider for example instance `hanoi`. The MILP model obtained with 14 linearization points (7 for each direction i to j and j to i) is solved to optimality by [71] in around 40 CPU minutes, and the solution has value 6,170,269. Such a solution is worse than the one obtained with the MINLP approach (6,109,620.90), and it is slightly NLP infeasible once the corresponding set of diameters is given to the NLP solver `Ipopt`. In addition, we fixed the diameters corresponding to the MINLP solution into the MILP model and realized that the solution is indeed infeasible for such a rough approximation. In fact, this set of diameters becomes feasible only using at least 170 (!!) linearization points. However, as mentioned above, solving the complete MILP model obtained using 170 points is out of the question even for a small-/medium-size instance as `hanoi`.

The trend outlined above is confirmed on the other 8 instances in our data set. Going from the smallest to the biggest, the only instance for which the MILP approach is effective is `shamir` which is small and easy enough to be solved with good accuracy and quality by using 14 linearization points; the value of 419,000 is proven to be optimal in a few seconds. For `blacksburg`, which is still pretty small, we used 30 linearization points, and the first feasible solution (with the discussed approximation) is obtained after 2 hours of CPU time. The best solution found within a time limit of 48 CPU hours has value 129,280.60 , which is larger than the best MINLP solution of value 118,251.09 . Approximating this instance seems to be rather hard; the MINLP diameter set is not feasible for the MILP model even allowing 4,000 linearization points. For `New York`, we used 30 linearization points, and the first feasible solution is obtained after 3 minutes, but its value is quite bad (61,382,605.6). After 2 CPU hours, the best solution value is 43,317,003.3 which becomes 40,474,098.3 after 2 days. Anyway the optimal solution found with the MINLP approach is not feasible for the MILP approximation considering less than 90 linearization points. If we run the MILP model with 90 linearization points, we are able to find the first feasible solution after more than 20 minutes (1,479 seconds, value 65,819,089.9), but after 3 hours we have still a quite bad solution (value 46,045,781.3).

For the `fossolo` set, even with very few linearization points, the MILP approach is unable to find feasible solutions within 2 days. A very inaccurate solution of bad quality has been found for `foss_poly_0` with only 6 linearization points (see [30]). Unfortunately, even providing the MILP model with the set of diameters found by the MINLP approach, no feasible solution can be obtained even allowing 1,000 linearization points.

Finally, for instances `pescara` and `modena`, which are the largest ones, no feasible solutions were obtained, independent of the number of linearization points.

7.6 Conclusions

In this chapter we have been able to get effective solutions, both in terms of quality and accuracy, to practical instances of water-network optimization problems. Although Mixed Integer Linear Programming models were known since the 80's for the problem, those models are very difficult to solve by sophisticated MILP solvers because they are somehow unnatural. A much more natural Mixed Integer Non Linear Programming formulation allowed us to find the above mentioned good solutions in very reasonable computing times. This success was achieved in two stages:

1. In a first phase, we could obtain from reasonable to good results with very low development time mainly because of the availability of software for finding good solutions

to MINLP problems and the easy interface to such software via the modeling language AMPL.

2. In a second phase, we moved to a more sophisticated analysis of both the model and the algorithm, and we have been able to improve over the initial results significantly by using special-purpose modeling tricks and by contributing to the open-source platform provided by the software `Bonmin` with effective adaptations to deal with non-convex MINLPs and multiple objective functions. The code developed in this context is committed to the `Bonmin v. trunk` repository for further use in different applications.

Our belief is that such a success can be obtained within the same framework for other instances of optimization problems having significant discrete and non-linear aspects.

Acknowledgments

We thank Pierre Bonami and Andreas Wächter for helping us to use `Bonmin` in the best possible way. The last two authors are partially supported by "Ministero dell'Università e della Ricerca" (MIUR), Italy. We thank two anonymous referees for a careful reading and very useful remarks. Finally, we thank Stefan Vigerske for interesting discussions on the subject.

Part IV

Tools for MINLP

Chapter 8

Tools for Mixed Integer Non-Linear Programming

This chapter is devoted to a brief review of available tools useful to solve Mixed Integer Non-Linear Programming. Following Chapter 1, we start reviewing the solvers addressed to special classes of MINLP: Mixed Integer Linear Programming problems in Section 8.1 and solvers for Non-Linear Programming problems in Section 8.2. In these two cases, only aspects closely related to MINLP problems are presented. On the other hand, the main focus is on MINLPs (Section 8.3). We devote Section 8.4 to the description of NEOS, a Server for Optimization [96]. In Section 8.5 we point out some difficulties arising from handling non-linear functions within an optimization model and how modeling languages help to overcome some of them. Finally, in Section 8.6 the most used libraries of MINLP problems are presented. Benchmark results on optimization solvers can be found at the web sites: `http://plato.asu.edu/bench.html` and `http://www.gamsworld.org`. They are not reported in this chapter, because they do not represent the focus of this Ph.D. thesis.

8.1 Mixed Integer Linear Programming solvers

In this section, few considerations, always limited to the topic of the thesis, about MILP solvers are presented. In Chapters 1 and 4 we described the standard techniques to approximate special classes of MINLP problems with an MILP model. As discussed in Chapter 4, piecewise linear approximation is used when non-linear functions are univariate and, in this case, Special Ordered Sets (SOS) can be defined. In some of the available MILP solvers, the definition of Special Ordered Sets is recognized and these sets of variables are treated in an ad-hoc way, defining special branching rules that usually positively influence the performance of the solver. Examples of these MILP solvers are the commercial solvers Cplex [71], Xpress [130] and the open-source solver Cbc [36]. However, it is possible to use any kind of MILP solver and define a piecewise linear function adding binary variables as explained in Chapter 4. In this case no special branching rule is used within the MILP solvers. Another important role that MILP solvers play in the solution of MINLP problems is that some MINLP solvers use substructures and algorithms from MILP. In addition, MILP solvers are also employed for solving MILP subproblems, when needed. These dependencies are described in Sections 8.3. We end the section with a consideration about the files which represent a MILP model. There are few standard file formats widely used by the most common MILP solvers. These are, for example, `mps` and `lp` files. No standard file format is available for models in which non-linear functions are involved. We discuss this issue in Sections 8.3 and 8.5.

8.2 Non-Linear Programming solvers

As extensively discussed in Section 1.3 and 1.4, also Non-Linear Programming subproblems play a fundamental role in the solution of Mixed Integer Non-Linear Programming problems. For this reason, this section is devoted to considerations and comments concerning NLP solvers. As explained in Section 1.2, different approaches are possible to the solution of NLPs. Thus, different types of solvers are available, commercial or open-source. Examples of the most widely used solvers are filterSQP, Ipopt, Knitro, Lancelot, Loqo, Minos, Pennon, Snopt. For a (not very recent, actually) survey on Non-Linear Programming software, we refer the reader to Nash [94]. Moreover, in [97] it is possible to find, at the end of each chapter describing an NLP method, some considerations about the software in which those techniques are implemented.

Usually NLP solvers have in common the following issue: they need the (first and usually second) derivatives of the non-linear functions involved in the model. Thus, the non-linear functions are assumed to be smooth. The necessity of derivatives is also the cause of the lack of a standard file format to express NLP/MINLP problems. The user needs to provide the NLP solver procedures which calculate the value of the non-linear function, the first and, possibly, the second derivative at a given point. This process is not straightforward for not expert users and could easily be source of errors. The introduction of modeling languages (see Section 8.5) helped to partially overcome such difficulties. In particular, they provide a unique, intuitive and flexible environment to express both linear and non-linear models. These frameworks are interfaced to most of the modern solvers and provide a tool for computing the approximation of the derivatives required in the NLP/MINLP case. For an introduction on Derivative Free Optimization, the discipline aimed at studing ad-hoc methods for problems for which the information about derivatives of the functions involved cannot be computed, the redear is referred to the book by Conn, Scheinberg and Vicente [7].

The last consideration concerns the role of NLP solvers within a MINLP method. The global optimality guarantee of some of the algorithms proposed for MINLPs depends also on the global optimality guarantee of the solution provided by the NLP solver. For example, consider the Branch-and-Bound (see Section 1.3). If, at each node, the corresponding NLP relaxation is solved to global optimality, then the method is exact not only for convex MINLPs, but also for non-convex MINLPs. However, as mentioned in Chapter 1, solving a non-convex NLP to global optimality is a hard task, thus usually NLP solvers guarantee only local optimality, if the NLP problem is non-convex. For this reason, specific techniques to solve non-convex MINLPs are proposed (see Section 1.4), as well as specific non-convex MINLP solvers (see the next section).

8.3 Mixed Integer Non-Linear Programming solvers

This section is devoted to software for Mixed Integer Non-Linear Programming. After some considerations about the solvers in general, we report information about the most widely used general-purpose MINLP solvers. One page is devoted to each solver. We do not describe special-purpose solvers, such as, for example, solvers for Semi-Definite Programming, Mixed

Integer Quadratic Programming, but only general-purpose MINLP solvers. For each solver, we give a brief summary of its characteristics (taken from manuals or web pages of the solvers) and a schematic table containing the following fields:

> Algorithm implemented: the algorithm implemented within the solver. A description of the main algorithms used in MINLP can be found in Sections 1.3 and 1.4. The most common algorithms are Branch-and-Bound (standard or spatial), Outer Approximation, Branch-and-Cut. If present, special tools, aimed at speeding-up the solver, are mentioned, for example bound reduction techniques.
>
> Type of algorithm: the algorithm implemented is exact/heuristic and under which assumptions (for example, convexity of the MINLP).
>
> Available Interface: how the user can interface to the solver. The most common possibilities are: (i) the solver is available as stand-alone executable, which is provided with the description of the model into a file; (ii) the user can provide information about the MINLP problem to be solved implementing some routines linked to the solver libraries; (iii) the user can write the model under a modeling language environment which is directly linked to the solver and call the solver from the environment (see Section 8.5 for details on modeling languages).
>
> Open-source software: only if the software is open-source, the source code is available and can be modified by the user.
>
> Programming Language: the programming language used to implement the software. This information is given only if the solver is open-source.
>
> Author: the authors of the software.
>
> Web site: the web site of the solver, where information and the user manual can be found.
>
> Dependencies: the external software which the solver depends on, i.e. if the solver uses (parts of) other software to exploit capabilities of, for example, interfacing to specific environments, solve special classes of MINLP, like NLP/MILP subproblems, or generating cuts. Often, as described in Chapter 1, algorithms for MINLP combines MILP and NLP techniques, for this reason solvers for these subproblems can be useful within a MINLP solver framework.

The reported data are useful also to understand the characteristics of the MINLP solvers in general.

The first consideration is that, as mentioned for NLPs, at the moment there is no standard file format for describing MINLP models. The reasons are the same explained in Section 8.2, because, also in the MINLP case, there are difficulties in expressing non-linear functions (as in NLP), and MINLP techniques often rely on NLP methods. Also in this case, these issues are partially overcome by modeling languages, see Section 8.5. A useful tool for the convertion of files within the most common files format can be found at the address (provided by Gams World) `http://www.gamsworld.org/performance/paver/convert_submit.htm`.

The second consideration concerns the type of algorithm: in particular, sometimes assumptions on the characteristics of the MINLP problems are provided to the solver. Some of the algorithms implemented are studied for convex MINLP and do not work with non-convex MINLPs. In other cases, methods studied for convex MINLPs can be used for non-convex MINLPs as heuristic algorithms, i.e. the solution provided by the solver is not guaranteed to be the global optimum. The last case is that the algorithm implemented is exact for non-convex MINLPs, so the software is a Global Optimization solver. In this case, the solution provided is guaranteed to be a global optimum. Clearly, methods of this type are much more complicated and their performance is, in general, worse than methods for convex MINLP. Thus, Global Optimization solvers are suitable for medium size problems. An example of the complexity of the Global Optimization methods is given by the large employment of complicated structures like symbolic expression trees, introduced in [118]. Methods such as spatial BB rely on the automatic definition of a convex relaxation of the MINLP problem. This can be defined in different ways, but the needing of reformulating the problem into a "standard form" is common to almost all these techniques. So, the first step preformed by these methods is reformulating the MINLP problem into a form which is "tractable", i.e. convex relaxations of each part of the reformulated model are well-known. This is obtained by adding "auxiliary variables" which "absorb" the non-linearities of the original constraints. In this way, complex non-linearities are subdivided into simpler non-linearities, at the cost of additional variables and constraints. The basic non-linearities are then relaxed using common envelopes/underestimators in a standard way. For a detailed description of the issues arising in developing and implementing a Global Optimization algorithm of such a type the reader is refereed to Liberti [84] or to the documentation of each specific solver.

The last consideration is about dependencies: as explained in Chapter 1, methods for MINLP are usually based on decomposition/relaxation of the problem into simpler subproblems which are typically NLP and MILP problems. For this reason, often fundamental ingredients of MINLP solver are reliable and fast NLP and MILP solvers. The large improvements on both NLP and MILP software sides highly influence general-purpose software for MINLPs and the recent success MINLP solvers in solving real-world applications is also due to them. Also for this reason, the kind of problems which the MINLP solvers are suitable to depends on the NLP and MILP solver integrated as part of the solving phase. From a research viewpoint, the extension and combination of efficient methods for NLP and MILP to MINLP is usually successfull. From a development viewpoint, the use of available NLP and MILP solvers makes the work of the developer easier and allows to concentrate on aspects and issues typical of MINLP problems, exploiting more stable techniques for NLP and MILP problems (see, e.g., [22]).

8.3.1 Alpha-Ecp

Alpha-ECP is an MINLP solver based on the extended cutting plane method (ECP). The ECP method is an extension of Kelley's cutting plane (CP) algorithm which was originally given for convex NLP problems [72]. In Westerlund and Pettersson [126] the method was extended to convex MINLP problems and in Westerlund et al. [128] further extended to MINLP problems with pseudo-convex constraints. The method was further extended in Pörn and Westerlund [107] and Westerlund and Pörn [127] and the current version of the method converges to the global optimal solution for non-convex MINLP problems having a pseudo-convex objective function and pseudo-convex inequality constraints. The method requires the solution of a MILP subproblem in each iteration. The MILP subproblems may be solved to optimality, but can also be solved to feasibility or only to an integer relaxed solution in intermediate iterations. This makes the ECP algorithm efficient and easy to implement. In the present implementation, a commercial MILP solver, Cplex, is used to solve the MILP subproblems. The problems to be solved can be supplied as text files in an extended `lp` format or if the constraints cannot be given in explicit form, then they can be given as user supplied sub-programs.

Algorithm implemented: Extended Cutting Plane method.

Type of algorithm: exact for MINLPs with a pseudo-convex objective function and pseudo-convex inequality constraints.

Available Interface: graphic interface, definition of the problem using ecp files (an extended `lp` format where non-linear constraints can be written using FORTRAN-90 syntax).

Open-source software: no.

Author: T. Westerlund and K. Lundqvist.

Web site: `www.abo.fi/~twesterl/A-ECPManual.pdf`.

Dependencies: Cplex.

8.3.2 BARON

BARON is a computational system for facilitating the solution of non-convex optimization problems to global optimality. The Branch-and-Reduce Optimization Navigator derives its name from its combining interval analysis and duality in its reduce arsenal with enhanced Branch-and-Bound concepts as it winds its way through the hills and valleys of complex optimization problems in search of global solutions.

Algorithm implemented: Branch-and-Reduce.

Type of algorithm: exact.

Available Interface: AIMMS and GAMS.

Open-source software: no.

Author: M. Tawarmalani and N.V. Sahinidis.

Web site: `http://www.andrew.cmu.edu/user/ns1b/baron/baron.html`.

Dependencies: Osl/Cplex, Minos/Snopt.

8.3.3 BONMIN

BONMIN (Basic Open-source Non-linear Mixed INteger programming) is an open-source code for solving general MINLP problems. It is distributed on COIN-OR (`www.coin-or.org`) under the CPL (Common Public License). BONMIN is OSI Certified Open Source Software. There are several algorithmic choices that can be selected with BONMIN. B-BB is a NLP-based Branch-and-Bound algorithm, B-OA is an outer-approximation decomposition algorithm, B-QG is an implementation of Quesada and Grossmann's Branch-and-Cut algorithm, and B-Hyb is a hybrid outer-approximation based Branch-and-Cut algorithm. Some of the algorithmic choices require the ability to solve MILP problems and NLP problems. The default solvers for these are, respectively, the COIN-OR codes Cbc and Ipopt. To solve (heuristically) a problem with non-convex constraints, one should only use the Branch-and-Bound algorithm B-BB. A few options have been designed in BONMIN specifically to treat problems that do not have a convex continuous relaxation. First, in the context of non-convex problems, the NLP solver may find different local optima when started from different starting points. Two options allow for solving the root node or each node of the tree with a user-specified number of different randomly-chosen starting points, saving the best solution found. The function to generate a random starting point chooses a random point (uniformly) between the bounds provided for the variable. In particular, if there are some functions that cannot be evaluated at some points of the domain, it may pick such points, and so it is not robust in that respect. Second, since the solution given by the NLP solver does not truly give a lower bound, BONMIN allows for changing the fathoming rule to continue branching even if the solution value to the current node is worse than the best-known solution.

Algorithm implemented: Branch-and-Bound, Outer-Approximation, LP/NLP based Branch-and-Bound QG, Hybrid, heuristics.

Type of algorithm: exact for convex MINLPs.

Available Interface: AMPL, GAMS, stand-alone executable (reading `nl` files), invocation through C/C++ program.

Open-source software: yes.

Programming Language: C++.

Author: P. Bonami et al. (see the web page).

Web site: `https://projects.coin-or.org/Bonmin`.

Dependencies: Cbc/Cplex, Ipopt/filterSQP.

8.3.4 Couenne

Couenne (Convex Over and Under ENvelopes for Non-linear Estimation) is a Branch-and-Bound algorithm to solve MINLP problems. Couenne aims at finding global optima of non-convex MINLPs. It implements linearization, bound reduction, and branching methods within a Branch-and-Bound framework. Its main components are: (i) an expression library; (ii) separation of linearization cuts; (iii) branching rules; (iv) bound tightening methods. It is distributed on COIN-OR under the Common Public License (CPL). Couenne is OSI Certified Open Source Software.

Algorithm implemented: spatial Branch-and-Bound, bound reduction.

Type of algorithm: exact.

Available Interface: AMPL, stand-alone executable (reading `nl` files), invocation through C/C++ program.

Open-source software: yes.

Programming Language: C++.

Author: P. Belotti et al. (see the web page).

Web site: `https://projects.coin-or.org/Couenne`.

Dependencies: Cbc, Ipopt.

8.3.5 DICOPT

DICOPT is a program for solving MINLP problems that involve linear binary or integer variables and linear and non-linear continuous variables. DICOPT (DIscrete and Continuous OPTimizer) was developed by J. Viswanathan and Ignacio E. Grossmann at the Engineering Design Research Center (EDRC) at Carnegie Mellon University. The program is based on the extensions of the outer-approximation algorithm for the equality relaxation strategy. The MINLP algorithm inside DICOPT solves a series of NLP and MILP subproblems. These subproblems can be solved using any NLP or MILP solver that runs under GAMS. The solver can handle non-convexities, but it does not necessarily obtain the global optimum. The GAMS/DICOPT system has been designed with two main goals in mind: (i) to build on existing modeling concepts and to introduce a minimum of extensions to the existing modeling language and provide upward compatibility to ensure easy transition from existing modeling applications to non-linear mixed-integer formulations; (ii) to use existing optimizers to solve the DICOPT subproblems. This allows one to match the best algorithms to the problem at hand and guarantees that any new development and enhancements in the NLP and MILP solvers become automatically and immediate available to DICOPT.

Algorithm implemented: Outer Approximation, Equality Relaxation, Augmented Penalty.

Type of algorithm: exact for convex MINLPs.

Available Interface: GAMS.

Open-source software: no.

Author: J. Viswanathan and I.E. Grossmann.

Web site: `www.gams.com/dd/docs/solvers/dicopt.pdf`.

Dependencies: GAMS environment, MILP and NLP solver under GAMS.

8.3.6 FilMINT

FilMINT is a solver for convex MINLPs. The solver is based on the LP/NLP algorithm of Quesada and Grossmann [108]. FilMINT combines the MINTO Branch-and-Cut framework for MILP with filterSQP used to solve the non-linear programs that arise as subproblems in the algorithm. The MINTO framework allows to easily employ cutting planes, primal heuristics, and other well-known MILP enhancements for MINLPs. FilMINT offers new techniques for generating and managing linearizations that are shown to be efficient on a wide range of MINLPs.

Algorithm implemented: LP/NLP based Branch-and-Bound QG.

Type of algorithm: exact for convex MINLPs.

Available Interface: AMPL.

Open-source software: no.

Programming Language: C, C++, FORTRAN.

Author: K. Abhishek, S. Leyffer and J. Linderoth.

Web site: `www.mcs.anl.gov/~leyffer/papers/fm.pdf`.

Dependencies: MINTO, filterSQP.

8.3.7 LaGO

LaGO (Lagrangian Global Optimizer) is a Branch-and-Cut algorithm to solve MINLPs. A linear outer approximation is constructed from a convex relaxation of the problem. Since it does not require an algebraic representation of the problem, reformulation techniques for the construction of the convex relaxation cannot be applied, and sampling techniques are used in case of non-quadratic non-convex functions. The linear relaxation is further improved by mixed-integer-rounding cuts. Also box reduction techniques are applied to improve efficiency. It is assumed to have procedures for evaluating function values, gradients, and Hessians of the functions. The restriction to black-box functions has the advantage that LaGO can handle very general functions, but has the disadvantage that advanced reformulation and box reduction techniques cannot be used. Hence, when sampling methods are applied no deterministic global optimization is guaranteed.

Algorithm implemented: Branch-and-Cut (reformulation, relaxation, linear cut generator).

Type of algorithm: exact for MIQQPs and convex MINLPs.

Available Interface: AMPL, GAMS, stand-alone executable (reading `nl` files), invocation through C/C++ program.

Open-source software: yes.

Programming Language: C++.

Author: I. Nowak and S. Vigerske.

Web site: `https://projects.coin-or.org/LaGO`.

Dependencies: Ipopt, Cgl, Clp.

8.3.8 LINDOGlobal

GAMS/LINDOGlobal finds guaranteed globally optimal solutions to general non-linear problems with continuous and/or discrete variables. GAMS/LINDOGlobal supports most mathematical functions, including functions that are non-smooth, such as abs(x) and/or even discontinuous, such as oor(x). The LINDO global optimization procedure (GOP) employs Branch-and-Cut methods to break an NLP model down into a list of subproblems. Given appropriate tolerances, after a finite, though possibly large number of steps a solution provably global optimal to tolerances is returned. GAMS/LINDOGlobal can automatically linearize a number of non-linear relationships, such as max(x,y), through the addition of constraints and integer variables, so the transformed linearized model is mathematically equivalent to the original non-linear model. Keep in mind, however, that each of these strategies will require additional computation time. Thus, formulating models, so they are convex and contain a single extremum, is desirable. In order to decrease required computing power and time it is also possible to disable the global solver and use GAMS/LINDOGlobal like a regular non-linear solver. GAMS/LINDOGlobal has a multistart feature that restarts the standard (non-global) non-linear solver from a number of intelligently generated points. This allows the solver to find a number of locally optimal points and report the best one found. This alternative can be used when global optimization is costly.

Algorithm implemented: Branch-and-Cut.

Type of algorithm: exact and heuristic methods available.

Available Interface: GAMS.

Open-source software: no.

Author: LINDO Systems.

Web site: `http://www.gams.com/solvers/solvers.htm#LINDOGLOBAL`.

Dependencies: GAMS, Conopt.

8.3.9 MINLPBB

MINLPBB is a package of FORTRAN 77 subroutines for finding solutions to MINLP problems. The package implements the Branch-and-Bound method in a non-linear setting. The package must be used in conjunction with both filterSQP and bqpd. Problems are specified in the same way as for filterSQP (i.e. either via subroutines or CUTE or AMPL). The additional integer structure is specified using a vector to identify the indices of integer variables. The user can influence the choice of branching variable by providing priorities for the integer variables.

Algorithm implemented: Branch-and-Bound.

Type of algorithm: exact for convex MINLPs.

Available Interface: CUTE, AMPL.

Open-source software: no.

Author: S. Leyffer.

Web site: `www-unix.mcs.anl.gov/~leyffer/solvers.html`.

Dependencies: filterSQP, bqpd.

8.3.10 MINOPT

MINOPT is a comprehensive and flexible package for the solution of various types of optimization problems. It features both an advanced modeling language for the clear and concise representation of complex mathematical models as well as a robust algorithmic framework for the efficient solution of wide variety of mathematical programming problems. MINOPT is capable of handling the following model types: Linear Programs, Non-linear Programs, Mixed Integer Linear Programs, Mixed Integer Non-linear Programs, Non-linear Programs with Differential and Algebraic Constraints (NLP/DAE), Mixed Integer Non-linear Programs with Differential and Algebraic Constraints (MINLP/DAE), Optimal Control Problems (OCP), Mixed Integer Optimal Control Problems (MIOCP). The MINOPT algorithmic framework also has the following additional features: efficient integration and sensitivity analysis, ability to switch easily among various solvers, ability to fine tune the solution algorithms with an extensive list of options. MINOPT has connections to a number of solvers and is able to exploit the features, options and efficiency of the solvers. MINOPT provides the following algorithms for the solution of MINLPs: Generalized Benders Decomposition, Outer Approximation and variants (OA, OA/ER, OA/ER/AP), Generalized Cross Decomposition (GCD).

Algorithm implemented: Generalized Benders Decomposition, Outer Approximation, Generalized Cross Decomposition.

Type of algorithm: exact for convex MINLPs.

Available Interface: stand-alone executable, Minopt modeling language.

Open-source software: no.

Author: C.A. Schweiger and C.A. Floudas.

Web site: `http://titan.princeton.edu/MINOPT/`.

Dependencies: Cplex/LPsolve, Minos/NPsol/Snopt.

8.3.11 SBB

SBB is a GAMS solver for MINLP models. It is based on a combination of the standard Branch-and-Bound method known from Mixed Integer Linear Programming and some of the standard NLP solvers already supported by GAMS. Currently, SBB can use CONOPT, MINOS and SNOPT as solvers for subproblems. SBB supports all types of discrete variables supported by GAMS, including: Binary, Integer, Semicontinuous, Semiinteger, SOS1, SOS2. The Relaxed Mixed Integer Non-linear Programming (RMINLP) model is initially solved using the starting point provided by the modeler. SBB will stop immediately if the RMINLP model is unbounded or infeasible, or if it fails. If all discrete variables in the RMINLP model are integer, SBB will return this solution as the optimal integer solution. Otherwise, the current solution is stored and the Branch-and-Bound procedure will start. During the Branch-and-Bound process, the feasible region for the discrete variables is subdivided, and bounds on discrete variables are tightened to new integer values to cut off the current non-integer solutions. Each time a bound is tightened, a new, tighter NLP subproblem is solved starting from the optimal solution to the previous looser subproblem. The objective function values from the NLP subproblem is assumed to be lower bounds on the objective in the restricted feasible space (assuming minimization), even though the local optimum found by the NLP solver may not be a global optimum. If the NLP solver returns a Locally Infeasible status for a subproblem, it is usually assumed that there is no feasible solution to the subproblem, even though the infeasibility only has been determined locally. If the model is convex, these assumptions will be satisfied and SBB will provide correct bounds. If the model is not convex, the objective bounds may not be correct and better solutions may exist in other, unexplored parts of the search space.

Algorithm implemented: Branch-and-Bound.

Type of algorithm: exact for convex MINLPs.

Available Interface: GAMS.

Open-source software: no.

Author: ARKI Consulting and Development.

Web site: `http://www.gams.com/solvers/solvers.htm#SBB`.

Dependencies: GAMS, Conopt/Minos/Snopt under GAMS.

8.4 NEOS, a Server for Optimization

Most of the solvers described in the previous sections can be freely accessed on the Internet through NEOS (Network-Enabled Optimization System), Server for Optimization [96], maintained by the Optimization Technology Center of Northwestern University, the Argonne National Laboratory and University of Wisconsin. A user can submit an optimization problem to the server and obtain the solution and running time statistics using the preferred solver through different interfaces. In particular, three ways to interact with the server are available:

1. Web Interface: from the web site the user can submit a problem, typically written with the AMPL or GAMS modeling languages (see Section 8.5), and obtain results and statistics via email.

2. Client Interface: the user can communicate with the server using a client implementation with different programming languages such as Python, Perl, PHP, C, C++, Java, Ruby (see [96] and [129] for details).

3. Kestrel Interface: the user can submit a problem to the server and use the results within a modeling environment. In the client machine, an executable file is called in order to directly communicate with the server within the AMPL or GAMS environment (see `http://www-neos.mcs.anl.gov/neos/kestrel.html` for details).

The chance to access to different solvers is a great opportunity for the optimization community, both because a user can test the performance of different solvers and decide which one is most suitable for the specific problem and because some of the solvers available are commercial solvers that cannot be used in other ways by users who do not own a license. The uniformity of the interfaces available for the different solvers makes easy to switch from one solver to another without any additional work. Another advantage is that it is very easy to use, so it is suitable for users not expert in optimization.

8.5 Modeling languages

Modeling languages are specifically studied to express optimization models (LP, NLP, MILP, MINLP) with an intuitive syntax avoiding the user to implement software using programming languages. They translate a model from a form easy to read for the user to a form readable by the solvers. The most common and intuitive modeling languages are the algebraic modeling languages (AML), widely used in (Mixed Integer) Non-Linear Programming. Their syntax is similar to the mathematical notation and this makes the syntax particularly intuitive and easy to read for the user. It is also flexible, allowing the user to implement within the modeling language framework also complicated algorithms, and compact, thanks to the use of abstract entities like sets and indices, making also large instances easy to write. The basic steps the AMLs perform are: (i) read the model provided by the user; (ii) translate the model into the appropriate form understandable to the solver; (iii) call the solver; (iv) read the solution provided by the solver. Note that AMLs do not own solver capabilities and use external solvers as black-boxes, but provide standard interfaces to the most common solvers. Another important tool of AMLs is the non-linear interpreter: using symbolic and/or automatic differentiation, it provides the NLP solver the information about the evaluation of functions, first and second derivatives at a given point. This makes AMLs a fundamental tool for (Mixed Integer) Non-Linear Programming. Moreover, being able to handle a wide range of mathematical operators, AMLs limit the consequences of the lack of a standard format for optimization models containing non-linearities. The two most widely used algebraic modeling languages in MINLP are AMPL [55] and GAMS [31]. Both are commercial products, but an evaluation limited version for students is available. The topic of modeling languages in general has other very important and interesting aspects which are not presented here because they do not represent the focus of this Ph.D. thesis.

8.6 MINLP libraries of instances

Mixed Integer Non-Linear Programming is a relatively recent topic. In the most recent years libraries of instances to use for benchmark were developed. In the following sections some of these libraries are described.

8.6.1 CMU/IBM Library

This library of convex MINLP instances was developed by people from Carnegie Mellon University and IBM within the open-source MINLP Project. The instances are 41, all of them are available in AMPL `nl` format and all but 3 of them also in GAMS format. The instances can be found at the web site `http://egon.cheme.cmu.edu/ibm/page.htm`. Useful information about the instances can be found in Bonami et al. [20].

8.6.2 MacMINLP Library

The MacMINLP library is maintained by Sven Leyffer and it is available at the web site `http://www.mcs.anl.gov/~leyffer/macminlp/index.html`. The library consists of 51 instances, 15 of which are convex (the information about the convexity of the instance is explicitally provided on the web site). The web site provides also the number of integer variables, the objective value of NLP relaxation and the optimal objective value or best solution found. The format of the instances is the `nl` file or the files `mod`, `dat` and `int` of the AMPL modeling language. Other available pieces of information are the ones provided by the so-called CUTE classification
(`http://www.rl.ac.uk/departments/ccd/numerical/cute/classification.html`),
such as the type of the objective function and the constraints, the smoothness of the problem, the origin and/or interest of the problem, the number of variables and constraints of the problem.

8.6.3 MINLPlib

The library, available at the web site `http://www.gamsworld.org/minlp/minlplib.htm`, consists of 266 instances. Provided information about the problems are the number of equations, variables, discrete variables, number of non-zeros, number of non-linear non-zeros, the best solution value and the reference to the origin of the instance. No information about the convexity of the instances is provided. Instances are available in GAMS format, but the web site provides also a Translation Service into other formats such as, for example, AMPL, BARON, GAMS, LINGO, MINOPT. More information about the instances can be found in Bussieck et al. [32]; the first author is also the coordinator of the library. In the following we report a (non complete) list of instances from MINLPlib which are convex (Table 8.1) and non-convex (Table 8.2). The collected pieces of information are taken from Nowak [98] and

Table 8.1: Convex instances of MINLPlib (info heuristically computed with LaGO).

alan	fo7_ar2_1	fo8	m7_ar25_1	nvs03	o7_ar5_1	st_miqp5	st_testph4
batch	fo7_ar25_1	fo9_ar2_1	m7_ar3_1	nvs10	o7	stockcycle	synthes1
csched2	fo7_ar3_1	fo9_ar25_1	m7_ar4_1	nvs11	o8_ar4_1	st_test1	synthes2
du-opt5	fo7_ar4_1	fo9_ar3_1	m7_ar5_1	nvs12	o9_ar4_1	st_test2	synthes3
du-opt	fo7_ar5_1	fo9_ar4_1	m7	nvs15	ravem	st_test3	tls12
ex1223a	fo7	fo9_ar5_1	meanvarx	nvs20	risk2bpb	st_test4	tls2
ex1223b	fo8_ar2_1	fo9	no7_ar2_1	o7_2	st_e14	st_test5	tls4
ex1223	fo8_ar25_1	gbd	no7_ar25_1	o7_ar2_1	st_miqp1	st_test6	tls5
fac1	fo8_ar3_1	m3	no7_ar3_1	o7_ar25_1	st_miqp2	st_test8	tls6
fac3	fo8_ar4_1	m6	no7_ar4_1	o7_ar3_1	st_miqp3	st_testgr1	tls7
fo7_2	fo8_ar5_1	m7_ar2_1	no7_ar5_1	o7_ar4_1	st_miqp4	st_testgr3	

Table 8.2: Non-convex instances of MINLPlib (info heuristically computed with LaGO).

batchdes	ex1222	feedtray2	nuclear14	nvs09	pump	super3
cecil_13	ex1224	feedtray	nuclear24a	nvs13	qap	super3t
chp_partload	ex1225	fuel	nuclear24b	nvs14	qapw	synheat
csched1	ex1226	fuzzy	nuclear24	nvs16	ravempb	tln12
deb10	ex1233	gasnet	nuclear25a	nvs17	saa_2	tln2
deb6	ex1243	gastrans	nuclear25b	nvs18	sep1	tln4
deb7	ex1244	gear2	nuclear25	nvs19	space25a	tln5
deb8	ex1252a	gear3	nuclear49a	nvs21	space25	tln6
deb9	ex1252	gear4	nuclear49	nvs23	space960	tln7
detf1	ex1263a	gear	nuclearva	nvs24	spectra2	tloss
eg_all_s	ex1263	gkocis	nuclearvb	oaer	spring	tltr
eg_disc2_s	ex1264a	hmittelman	nuclearvc	oil2	st_e13	uselinear
eg_disc_s	ex1264	johnall	nuclearvd	oil	st_e15	util
eg_int_s	ex1265a	lop97ic	nuclearve	ortez	st_e27	var_con10
elf	ex1265	lop97icx	nuclearvf	parallel	st_e29	var_con5
eniplac	ex1266a	mbtd	nvs01	prob02	st_e31	waste
enpro48	ex1266	nous1	nvs02	prob03	st_e36	water4
enpro48pb	ex3	nous2	nvs04	prob10	st_e38	waterx
enpro56	ex3pb	nuclear10a	nvs06	procsel	st_e40	waterz
enpro56pb	ex4	nuclear14a	nvs07	product2	super1	windfac
ex1221	fac2	nuclear14b	nvs08	product	super2	

integrated with the use of a feature of LaGO which heuristically tries to assert the convexity of the instance. LaGO does it by sampling the Hessian and computing eigenvalues of the MINLP problem. Note that the tool does not provide an exact information, i.e. in few cases it can be not correct. Moreover, LaGO checks only if every constraint function is convex, concave, or neither of both. If functions are non-convex, but the feasible region is convex, the model is considered non-convex. Thus, the information collected in Tables 8.1) and 8.2 are not to be considered exact. [1]

[1]Thanks to Stefan Vigerske for useful comments and discussions about LaGO.

Bibliography

[1] K. Abhishek. *Topics in Mixed Integer Nonlinear Programming.* PhD thesis, Lehigh University, 2008.

[2] T. Achterberg. *Constraint Integer Programming.* PhD thesis, Technische Universitt Berlin, 2007.

[3] T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4:77–86, 2007.

[4] T. Achterberg, T. Koch, and A. Martin. MIPLIB 2003. *Operations Research Letters*, 34:1–12, 2006. See `http://miplib.zib.de`.

[5] C.S. Adjiman, I.P. Androulakis, and C.A. Floudas. Global optimization of minlp problems in process synthesis and design. *Computers and Chemical Engineering*, 21:445–450, 1997.

[6] 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.

[7] L.N. Vicente A.R. Conn, K. Scheinberg. *Introduction to derivative free Optimization.* MPS/SIAM Book Series on Optimization, SIAM, Philadelphia, 2008.

[8] S. Artina, C. Bragalli, A. Lodi, and P. Toth. Approccio MILP al problema di optimal design di reti di distribuzione idrica. In *Atti del 28° Congresso Nazionale di Idraulica e Costruzioni Idrauliche (in Italian)*, volume 1, pages 67–70, 2002.

[9] S. Artina and J. Walker. Sull'uso della programmazione a valori misti nel dimensionamento di costo minimo di reti in pressione. In *Atti dell'Accademia delle Scienze dell'Istituto di Bologna (in Italian)*, Anno 271, Serie III, Tomo X, 1983.

[10] D. A. Babayev. Piece-wise linear approximation of functions of two variables. *Journal of Heuristics*, 2:313–320, 1997.

[11] A. Baillo, M. Ventosa, A. Ramos, M. Rivier, and A. Canseco. Strategic unit commitment for generation companies in deregulated electricity markets. In *Proceedings of the 1999 DIMACS/EPRI Workshop*, 1999.

[12] M.S. Bazaraa and C.M. Shetty. *Nonlinear Programming. Theory and algorithms.* John Wiley and Sons, New York, 1979.

[13] E.M.L. Beale and J.A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In J. Lawrence, editor, *OR 69. Proceedings of the Fifth International Conference on Operational Research*, pages 447–454. Tavistock Publications, 1970.

[14] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex minlp. Technical report, IBM Research Report RC24620, 2008.

[15] J. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:267–299, 1962.

[16] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. *Discrete Optimization*, 4:63–76, 2007.

[17] D.P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 1999.

[18] D. Bertsimas and J.N. Tsitsiklis. *Introduction to Linear Optimization.* Dynamic Ideas and Athena Scientific, Belmont, Massachusetts, March, 2008.

[19] D. Bertsimas and R. Weismantel. *Optimization over Integers.* Dynamic Ideas, Belmont, Massachusetts, January, 2005.

[20] P. Bonami, L.T. Biegler, A.R. Conn, G. Cornuéjols, I.E. Grossmann, C.D. Laird, J. Lee, A. Lodi, F. Margot, N. Sawaya, and A. Wächter. An algorithmic framework for convex mixed integer nonlinear programs. *Discrete Optimization*, 5:186–204, 2008.

[21] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119:331–352, 2009.

[22] P. Bonami, J. Forrest, J. Lee, and A. Wäcther. Rapid development of an minlp solver with coin-or. *Optima*, 75:1–5, December, 2007.

[23] P. Bonami and J.P.M. Goncalves. Primal heuristics for mixed integer nonlinear programs. Technical report, IBM Research Report RC24639, 2008.

[24] P. Bonami and J. Lee. Bonmin users' manual. Technical report, June 2006.

[25] P. Bonami and M. Lejeune. An exact solution approach for portfolio optimization problems under stochastic and integer constraints. *Operations Research*, To appear.

[26] Bonmin. `projects.coin-or.org/Bonmin`, v. 1.0.1.

[27] Bonmin. `projects.coin-or.org/Bonmin`, v. trunk.

[28] A. Borghetti, C. D'Ambrosio, A. Lodi, and S. Martello. An milp approach for short-term hydro scheduling and unit commitment with head-dependent reservoir. *IEEE Transactions on Power Systems*, 23:1115–1124, 2008.

[29] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[30] C. Bragalli, C. D'Ambrosio, J. Lee, A. Lodi, and P. Toth. An MINLP solution method for a water network problem. In Y. Azar and T. Erlebach, editors, *Algorithms – ESA 2006*, volume 4168 of *Lecture Notes in Computer Science*, pages 696–707. Springer–Verlag, Berlin Heidelberg, 2006.

[31] A. Brooke, D. Kendrick, and A. Meeraus. Gams: A user's guide, 1992.

[32] M.R. Bussieck, A.S. Drud, and A. Meeraus. Minlplib - a collection of test models for mixed-integer nonlinear programming. *INFORMS Journal on Computing*, 15:114–119, 2003.

[33] R. Byrd, J. Nocedal, and R. Waltz. Knitro: An integrated package for nonlinear optimization. In G. Di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, pages 35–60. Springer, 2006.

[34] M. Carrión and J. M. Arroyo. A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems*, 21:1371–1378, 2006.

[35] J.P.S. Catalão, S.J.P.S. Mariano, V.M.F. Mendes, and L.A.F.M. Ferreira. Parameterisation effect on the behaviour of a head-dependent hydro chain using a nonlinear model. *Electric Power System Research*, 76:404–412, 2006.

[36] Cbc. `https://projects.coin-or.org/Cbc`.

[37] C.W. Chang and J.G. Waight. A mixed integer linear programming based hydro unit commitment. In *Power Engineering Society Summer Meeting*, 1999.

[38] G.W. Chang, M. Aganagic, J.G. Waight, J. Medina, T. Burton, S. Reeves, and M. Christoforidis. Experiences with mixed integer linear programming based approaches on short-term hydro scheduling. *IEEE Transactions on Power Systems*, 16(4):743–749, 2001.

[39] A.J. Conejo, J.M. Arroyo, J. Contreras, and F.A. Villamor. Self-scheduling of a hydro producer in a pool-based electricity market. *IEEE Transactions on Power Systems*, 17(2):1265–1272, 2002.

[40] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming B*, 112:3–44, 2008.

[41] M. Cunha and J. Sousa. Water distribution network design optimization: Simulated annealing approach. *Journal of Water Resources Planning and Management, ASCE*, 125:215–221, 1999.

[42] G.C. Dandy, A.R. Simpson, and L.J. Murphy. An improved genetic algorithm for pipe network optimization. *Water Resources Research*, 32:449–458, 1996.

[43] E. Danna, E. Rothberg, and C. Le Pape. Exploiting relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102:71–90, 2005.

[44] G.B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[45] M. Duran and I.E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[46] J. Eckstein and M. Nediak. Pivot, cut, and dive: a heuristic for 0-1 mixed integer programming. *Journal of Heuristics*, 13:471–503, 2007.

[47] G. Eiger, U. Shamir, and A. Ben-Tal. Optimal design of water distribution networks. *Water Resources Research*, 30:2637–2646, 1994.

[48] EPANET. `www.epa.gov/ORD/NRMRL/wswrd/epanet.html`, v. 2.0.

[49] E.O. Finardi, E.L. Da Silva, and C. Sagastizabal. Solving the unit commitment problem of hydropower plants via lagrangian relaxation and sequential quadratic programming. *Computational and Applied Mathematics*, 24(3):317–341, 2005.

[50] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2004.

[51] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2002.

[52] M. Fischetti and A. Lodi. Optimizing over the rst chvátal closure. *Mathematical Programming*, 110:3–20, 2007.

[53] M. Fischetti and D. Salvagnin. Feasibility pump 2.0. Technical report, ARRIVAL project, October 2008.

[54] R. Fletcher. *Practical Method of Optimization.* John Wiley and Son, 2000.

[55] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming.* Duxbury Press/Brooks/Cole Publishing Co., second edition, 2003.

[56] A. Frangioni and C. Gentile. Perspective cuts for a class of convex 0-1 mixed integer programs. *Mathematical Programming*, 106:225–236, 2006.

[57] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[58] O. Fujiwara and D.B. Khang. A two-phase decomposition method for optimal design of looped water distribution networks. *Water Resources Research*, 26:539–549, 1990.

[59] J. García-González and G.A. Castro. Short-term hydro scheduling with cascaded and head-dependent reservoirs based on mixed-integer linear programming. In *Power Tech Proceedings, 2001 IEEE Porto*, 2001.

[60] J. García-González, E. Parrilla, and A. Mateo. Risk-averse profit-based optimal scheduling of a hydro-chain in the day-ahead electricity market. *European Journal of Operational Research*, 181(3):1354–1369, 2007.

[61] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[62] A.M. Geoffrion. Generalized benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[63] F.W. Glover and G.A. Kochenberger, editors. *Handbook of Metaheuristics.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 2003.

[64] R.E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[65] I.E. Grossmann. Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and Engineering*, 3:227–252, September, 2002.

[66] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, Berlin Heidelberg, 1988.

[67] W. Grubauer, R. Unverwood, M. Telgenhoff, and R. Frowd. Optimal hydro generation and interchange scheduling with flow-dependent channel constraints. In *Proceeding of Energy Management and Power Delivery (EMPD '95)*, 1995.

[68] O. Günlük, J. Lee, and R. Weismantel. Minlp strengthening for separable convex quadratic transportation-cost ufl. Technical report, IBM Research Report RC24213, 2007.

[69] O. Günlük and J. Linderoth. Perspective relaxation of mixed integer nonlinear programs with indicator variables. In A. Panconesi A. Lodi and G. Rinaldi, editors, *IPCO, Lecture Notes in Computer Science*, pages 1–16. Springer, 2008.

[70] O.K. Gupta and V. Ravindran. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[71] Ilog-Cplex. `www.ilog.com/products/cplex`, v. 10.1.

[72] J.E. Kelley Jr. The cutting-plane method for solving convex programs. *Journal of SIAM*, 8:703–712, 1960.

[73] R. Karuppiah and I.E. Grossmann. A lagrangean based branch-and-cut algorithm for global optimization of nonconvex mixed-integer nonlinear programs with decomposable structures. *Journal of Global Optimization*, 41:163–186, 2008.

[74] W. Karush. *Minima of functions of several variables with inequalities as side constraints.* PhD thesis, Master's thesis, 1939.

[75] A.B. Keha, I.R. de Farias, and G.L. Nemhauser. Models for representing piecewise linear cost functions. *Operations Research Letters*, 32:44–48, 2004.

[76] P. Kesavan and P.I. Barto. Generalized branch-and-cut framework for mixed-integer nonlinear optimization problems. *Computers and Chemical Engineering*, 24:1361–1366, 2000.

[77] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the 2nd Berkley Symposium on Mathematical Statistics and Probability*, pages 481–493. University Press, Berkley, California, 1951.

[78] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28:497–530, 1960.

[79] K.E. Lansey and L.W. Mays. Optimization model for water distribution system design. *Journal of Hydraulic Engineering*, 115:1401–1418, 1989.

[80] J. Lee and D. Wilson. Polyhedral methods for piecewise-linear functions i: the lambda method. *Discrete Applied Mathematics*, 108:269–285, 2001.

[81] S. Leyffer. Integrating sqp and branch-and-bound for mixed integer nonlinear programming. *Computational Optimization and Applications*, 18:295–309, 2001.

[82] S. Leyffer. User manual for `MINLP_BB`. Technical report, University of Dundee, April 1998; revised March 1999.

[83] L. Liberti. *Reformulation and Convex Relaxation Techniques for Global Optimization*. PhD thesis, Imperial College London, UK, 2004.

[84] L. Liberti. Writing global optimization software. In L. Liberti and N. Maculan, editors, *Global Optimization: from Theory to Implementation*, pages 211–262. Springer, Berlin, 2006.

[85] L. Liberti, S. Cafieri, and F. Tarissan. Reformulations in mathematical programming: A computational approach. In A.E. Hassanien, A. Abraham, F. Herrera, W. Pedrycz, P. Siarry A. Carvalho, and A. Engelbrecht, editors, *Foundations on Computational Intelligence, Studies in Computational Intelligence*. Springer, Berlin, to appear.

[86] L. Liberti, G. Nannicini, and N. Mladenovic. A good recipe for solving minlps. In *Matheuristics08 Proceedings*. 2008.

[87] A. Lodi. Mip computation and beyond. In M. Junger, T. Liebling, D. Naddef, W. Pulleyblank, G. Reinelt, G. Rinaldi, and L. Wolsey, editors, *50 Years of Integer Programming 1958-2008*. Springer Verlag, 2008.

[88] N. Lu, J.H. Chow, and A.A. Desrochers. Pumped-storage hydro-turbine bidding strategies in a competitive electricity market. *IEEE Transactions on Power Systems*, 19(2):834–841, 2004.

[89] A. Martin, M. Moller, and S. Moritz. Mixed integer models for the stationary case of gas network optimization. *Mathematical Programming*, 105:563–582, 2006.

[90] Mathematica. `www.wolfram.com/products/mathematica/index.html`, v. 7.0.

[91] Matlab. `http://www.mathworks.com/products/matlab/`, R2007a.

[92] G.P. McCormick. Computability of global solutions to factorable nonconvex programs: Part i - convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[93] G. Nannicini, P. Belotti, and L. Liberti. A local branching heuristic for minlps. *ArXiv, paper 0812.2188*, 2009.

[94] S.G. Nash. Software survey: Nonlinear programming. *OR/MS Today*, 25:36–38, June, 1998.

[95] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley and Sons, New York, 1988.

[96] NEOS. `www-neos.mcs.anl.gov/neos`, v. 5.0.

[97] J. Nocedal and S.J. Wright. *Numerical Optimization.* Springer Series in Operations Research, 2006.

[98] I. Nowak. *Relaxation and Decomposition Methods for Mixed Integer Nonlinear Programming.* International Series of Numerical Mathematics, Birkhäuser Verlag, 2005.

[99] I. Nowak, H. Alperin, and S. Vigerske. Lago – an object oriented library for solving minlps. In *Global Optimization and Constraint Sarisfaction*, volume 2861 of *Lecture Notes in Computer Science*, pages 32–42. Springer, Berlin Heidelberg, 2003.

[100] I. Nowak and S. Vigerske. Lago - a (heuristic) branch and cut algorithm for nonconvex minlps. *Central European Journal of Operations Research*, 16:127–138, 2008.

[101] S.O. Orero and M.R. Irving. A genetic algorithm modelling framework and solution technique fo short term optimal hydrothermal scheduling. *IEEE Transactions on Power Systems*, 13(2):501–518, 1998.

[102] M.W. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6:1–7, 1987.

[103] N.P. Padhy. Unit commitment - a bibliographical survey. *IEEE Transactions on Power Systems*, 19(2):1196–1205, 2004.

[104] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Prentice Hall, Englewood Cliffs, N.J., 1988.

[105] P.M. Pardalos and H.E. Romeijn. *Handbook of Global Optimization Volume 2.* Kluwer Academic Publishers, Netherlands, 2002.

[106] M.R. Piekutowki, T. Litwinowcz, and R.J. Frowd. Optimal short-term scheduling for a large-scale cascaded hydro system. *IEEE Transactions on Power Systems*, 9(2):805–811, 1994.

[107] R. Pörn and T. Westerlund. A cutting plane method for minimizing pseudo-convex functions in the mixed-integer case. *Computers and Chemical Engineering*, 24:2655–2665, 2000.

[108] I. Quesada and I.E. Grossmann. An lp/nlp based branch and bound algorithm for convex minlp optimization problems. *Computer and Chemical Engineering*, 16:937–947, 1992.

[109] F. Rendl, G. Rinaldi, and A. Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. Technical report, Alpen-Adria-Universitt Klagenfurt, Inst. f. Mathematik, 2008.

[110] H. Ryoo and N. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8:107–138, 1996.

[111] N.V. Sahinidis. Baron: a general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.

[112] D. A. Savic and G. A. Walters. Genetic algorithms for the least-cost design of water distribution networks. *ASCE Journal of Water Resources Planning and Management*, 123:67–77, 1997.

[113] A. Saxena, P. Bonami, and J. Lee. Disjunctive cuts for non-convex mixed integer quadratically constrained programs. In A. Panconesi A. Lodi and G. Rinaldi, editors, *IPCO, Lecture Notes in Computer Science*, pages 17–33. Springer, 2008.

[114] J. C. Jr. Schaake and D. Lai. Liner programming and dynamic programming application to water distribution network design. Report, Hydrodynamics Laboratory, Department of Civil Engineering, School of Enrineering, Massachusetts Institute of Technology, Cambridge, Massachussets, 1969.

[115] T.J. Scott and E.G. Read. Modelling hydro reservoir operation in a deregulated electricity market. *International Transactions in Operational Research*, 3(3/4):243–253, 1996.

[116] H. D. Sherali, S. Subramanian, and G. V. Loganathan. Effective relaxation and partitioning schemes for solving water distribution network design problems to global optimality. *Journal of Global Optimization*, 19:1–26, 2001.

[117] N.S. Sinha, R. Chakrabarti, and P.K. Chattopadhyay. Fast evolutionary programming techniques for short-term hydrothermal scheduling. *IEEE Transactions on Power Systems*, 18(1):214–220, 2003.

[118] E.M.B. Smith and C.C. Pantelides. A symbolic reformulation/spatial branch and bound algorithm for the global optimization of nonconvex minlps. *Computers and Chemical Engineering*, 23:457–478, 1999.

[119] M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.

[120] J.A. Tomlin. A suggested extension of special ordered sets to non-separable non-convex programming problems. In P. Hansen, editor, *Studies on Graphs and Discrete Programming*, pages 359–370. North-Holland Publishing Company, 1981.

[121] M. Van Den Boomen, A. Van Mazijk, and R.H.S. Beuken. First evaluation of new design concepts for self-cleaning distribution networks. *Journal of Water Supply: Research and Technology AQUA*, 53:43–50, 2004.

[122] J.P. Vielma and G.L. Nemhauser. Modeling disjunctive constraints with logaritmic – number of binary variables and constraints. In A. Lodi, A. Panconesi, and G. Rinaldi, editors, *IPCO, Lecture Notes in Computer Science*, pages 199–213. Springer, 2008.

[123] A. Wächter and L. T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.

[124] T.M. Walski. *Analysis of Water Distribution Systems.* Van Nostrand Reinhold Company, New York, N.Y., 1984.

[125] T.M. Walski, D.V. Chase, and D.A. Savic. *Water Distribution Modeling.* Haestad Methods, Inc., Waterbury, CT, U.S.A., 2001.

[126] T. Westerlund and F. Pettersson. A cutting plane method for solving convex minlp problems. *Computers and Chemical Engineering*, 19:S131–S136, 1995.

[127] T. Westerlund and R. Pörn. Solving pseudo-convex mixed integer problems by cutting plane techniques. *Optimization and Engineering*, 3:253–280, 2002.

[128] T. Westerlund, H. Skrifvars, I. Harjunkoski, and R. Pörn. An extended cutting plane method for solving a class of non-convex minlp problems. *Computers and Chemical Engineering*, 22:357–365, 1998.

[129] XML-RPC. `http://www.xmlrpc.com`.

[130] Xpress. `http://www.dashoptimization.com`.

[131] C. Xu and I.C. Goulter. Reliability-based optimal design of water distribution networks. *Journal of Water Resources Planning and Management*, 125:352–362, 1999.