Deterministic Methods for Mixed Integer Nonlinear Programming

Sven Leyffer
PhD Thesis
Department of Mathematics & Computer Science
University of Dundee
Dundee

December 1993

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to the Problem

Many optimization problems involve *integer* or *discrete variables* and can be modelled as Mixed Integer Nonlinear Programming problems (MINLPs). These variables can variously be integer variables modelling for example numbers of men, or zero–one variables modelling decisions, or discrete variables modelling, for example, equipment sizes. In addition to these there may also exist *continuous variables* which for example may represent pressures or temperatures. Nonlinearities come into the model either through physical properties such as enthalpy and vapour/liquid equilibrium, or may also involve the decision variables through for example economies of scale. The function to be optimized in this context usually involves costs and profits from the design.

Discrete variables occur whenever the variables model for example sizes which are taken from a prescribed finite domain. Amir and Hasegawa [1], for instance, give an example in which the total cost of a reinforced concrete beam is minimized subject to maximum load constraints. The problem is discrete since the steel reinforcements are available in discrete sizes only. A similar example is given by Sandgren [65], who minimizes the cost of a pressure vessel whose shell and heads are made from standard sized steel. The constraints reflect design specifications for the pressure vessel such as minimum thickness of the shell and minimum volume of the tank.

Integer variables often model the number of a certain commodity. Sandgren [65] reports an example where he attempts to find the number of teeth in a gear train to match a given design gear ratio in the l_2 sense. This results in a nonlinear and nonconvex objective function in four integer variables with simple bounds on the integer variables.

Binary variables play an important role in integer programming and are used widely in applications. They can be used to model decisions such as at which plate of a distillation column external feed enters (e.g. Viswanathan and Grossmann [73]). Binary variables are also very popular as a replacement for discrete or integer variables. In fact one could arguably restrict attention to binary variables, since they can replace both integer and discrete variables. However, the drawback of replacing integer or discrete variables lies in the increase in the number of variables. For every integer or discrete variable one binary variable has to be introduced for every possible integer/discrete value, so that an integer variable $0 \leq y \leq N$ is replaced by $N + 1$ binary variables. A special class of binary variables are Specially Ordered Sets (SOS) of type I and II. SOS I is a set of binary variables out of which exactly one member is non–zero. SOS II is a set of binary variables out of which at most two members which must be adjacent are non–zero (e.g. [78]).

Throughout the thesis no distinction will be made between the different types of integer or discrete variables referred to above. This does not constitute a great loss of generality if it is assumed that the underlying MILP or MIQP solver is capable of handling any type of discrete

variable. It is also possible to replace a discrete variable by an integer variable or by a number of zero–one variables. In the rest of the thesis the term integer variables is taken to include the possibility of discrete but non–integer variables.

All of the above cases are conveniently expressed in the following model problem

$$\mathrm{P}\left\{\begin{array}{ll}\min\limits_{x,y} & f(x,y)\\ \text{subject to} & g(x,y)\leq 0\\ & x\in X,\ y\in Y \text{ integer}\end{array}\right.$$

where f is the nonlinear objective function and g are the nonlinear constraints. The sets X and Y usually represent simple bounds on x and y respectively. The main assumptions on f, g, X, and Y are discussed in Section 1.5.

Problem P is one of the most difficult optimization problems. It falls into the class of $\mathcal{NP}$-complete problems. That means that no polynomial time algorithm is known for P and that if one existed, it would also be a polynomial time algorithm for any other problem in $\mathcal{NP}$ (see e.g. Nemhauser and Wolsey [56] Chapter I.5 and Schrijver [67]). In the following section a number of issues involved in MINLP problems are discussed.

1.2 Issues in MINLP

In this section the issues and problems that arise in MINLP are discussed by first examining two special cases which are embedded in P, namely MILP and NLP problems. MINLP problems possess also a number of features which are unique in the sense that they do not occur either in NLP or MILP problems and these are listed at the end of this section.

MILP problems are combinatorial optimization problems with an exponential number of integer feasible points. By fixing the integer variables and solving the resulting LP subproblem in the continuous variables only, it is comparatively easy to find a *local solution* to an MILP, that is a point that satisfies the first order Kuhn–Tucker conditions for a fixed assignment of integer variables. The number of these local solutions is exponential and since unlike in LP no optimality conditions exist the task of choosing the optimum among the local minima is very hard.

The lack of suitable optimality conditions for MILP implies that any MILP algorithm faces the double task of finding and *verifying* an optimal solution. Thus even though the algorithm were started at the global optimum it would still require a possibly exponential number of iterations to recognize the optimality. In the case of a branch–and–bound algorithm (described in Chapter 3) it is therefore usually necessary to examine further nodes in the tree once the optimum has been found before the optimality is verified.

A consequence of the combinatorial nature of the problem is that for most algorithms there exists a worst case example for which the algorithm has to solve an exponential number of subproblems. Jeroslow [40] gives a trivial class of integer problems for which any branch–and–bound algorithm has to expand an exponential number of nodes before it discovers that the problem is infeasible. His result is valid for a wide range of enumerative schemes and can be modified to give worst case behaviour for almost any other algorithm. These worst case examples agree with practical experience with integer programming algorithms which indicates an exponential growth in the computing time as the number of variables is increased. Thus, while MILP problems can be solved in a finite number of steps, this number grows usually exponentially in the number of variables.

On the other hand, LP and QP problems can be solved in polynomial time. For example, Karmarkar [41] gives a polynomial time algorithm for LP problems and Gill, Murray, Saunders, Tomlin and Wright [28] show that Karmarkar's algorithm is related to the logarithmic barrier function. However, many practical algorithms for LP and QP problems use the Simplex method or

an active set method. For those algorithms there exist worst case examples in which they visit a number of vertices which increases exponentially with the problem size, but this is not very likely to occur in practice. Thus a major difference between MILP and LP problems is that – in practice – the former require an exponential number of iterations while the latter can be solved in a number of iterations that is bounded by a polynomial in the problem size.

LP and QP problems form special classes of NLP problems. However, more general NLP problems cannot be solved in a finite number of steps and usually an iterative scheme has to be applied to solve them. Nevertheless, finding a local solution to an NLP problem is a much simpler task than finding a solution to an MILP problem, since methods like SQP (c.f. Section 2.4) ultimately exhibit a second order rate of convergence and the solver terminates once the first order conditions are satisfied to sufficient accuracy.

Another difference between MILP and NLP problems is that while every local solution of an MILP is a global solution, the same is only true for NLP problems under an additional convexity assumption (c.f. Section 2.3).

MINLP problems combine the two aspects of MILP and NLP problems but also have some features which are unique. While a strict convexity assumption ensures the global uniqueness of an NLP solution, the same does not hold for MINLP problems. Loh and Papalambros [52] give a very good illustration of such a situation which is given in Figure 1.1 in a slightly more general form. The picture shows the contours of a convex objective function and a nonlinear constraint which is indicated by the broken thick line and the two axes. The continuous feasible region lies between the constraint and the axis while the discrete feasible points are the dots that lie within the feasible region. The two discrete optima are indicated by circled dots and it is also shown where the unique continuous minimizer lies.

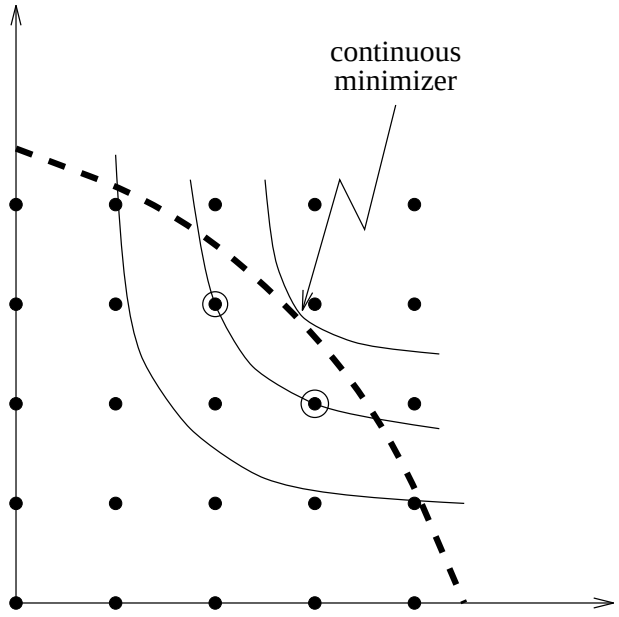

Figure 1.1: Multiple Optima for a strictly convex MINLP

It is possible to derive upper bounds on the difference between an optimal solution to an MILP problem and an optimal solution to its LP relaxation. However, these bounds are usually very weak since they involve the dimension of the problem and a constant that depends on the largest absolute subdeterminand of the integral coefficient matrix. Werman and Magagnosc [76] generalize

these bounds to MINLP problems with a separable and convex objective and linear constraints.

Having discussed some of the issues involved in MINLP programming, the next two sections provide an overview of methods for solving MINLP problems.

1.3 Deterministic Methods in MINLP

Methods for solving MINLP problems fall into two broad classes. The first class is formed by deterministic methods which – given enough time and provided the problem satisfies certain conditions such as convexity – terminate with a guaranteed solution or an indication that the problem has no integer solution. All deterministic methods have in common that they perform an exhaustive tree search with rules that enable them to limit the search to a subtree. Heuristic methods form the second class of methods. These methods do not provide a guarantee that on termination the incumbent is a minimizer. Any deterministic method that is applied to a problem which does not satisfy for instance a convexity assumption becomes a heuristic method. Deterministic methods are reviewed in this section while heuristic approaches are discussed in the next section.

There exists a very wide variety of deterministic methods for MILP problems (e.g. Nemhauser and Wolsey [56] and Schrijver [67]). However, only a few of them are applicable to MINLP problems. Among the general purpose MILP methods which do not generalize are implicit enumeration (e.g. Taha [69]), Gomory's cutting plane algorithm (e.g. Garfinkel and Nemhauser [24]) and dynamic programming (e.g. Greenberg [32]). The remainder of this section surveys those methods that can be applied to P. Among them are branch–and–bound ([5], [7], [9], [13], [17], [24], [32], [35], [36], [45], [47], [51], [56], [64], [65], [69], [75]), Benders Decomposition ([6], [20], [21], [22], [26], [48], [49], [62], [75]), outer approximation ([15], [16], [42] ,[43], [60], [73], [79]) and Lagrangean Decomposition([54]).

Branch–and–bound in particular has been very successful and underpins all the other methods. For instance, the master program relaxations of outer approximation, Benders Decomposition or Lagrangean Decomposition are all solved by a branch–and–bound routine. The first branch–and–bound algorithm was proposed by Land and Doig [47]. Little, Murty, Sweeney and Karel [51] applied the idea to the Travelling Salesman Problem. The variable dichotomy scheme which is now a standard was introduced by Dakin [13]. A short annotated bibliography is given in Nemhauser and Wolsey [56]. The main idea behind branch–and–bound is to solve continuous relaxations of the original problem and to divide the feasible region, eliminating the fractional solution of the relaxed problem. Continuing in this manner a tree of problems is created which is searched for the integer optimum. Dakin [13] was the first to realize that this scheme does not require linearity of the problem functions (see also Garfinkel and Nemhauser [24] or Taha [69]).

Gupta and Ravindran [35] implement a branch–and–bound routine for MINLP problems. They examine the effects of problem size on the solution time and suggest from an experimental study that the solution time increases linearly with the number of integer variables and is also proportional to the number of constraints in the model. This result is very surprising in view of the complexity of MINLP problems. Our own results, presented in Chapter 6 point to an increase in computing time which is exponential in the number of variables (e.g. the results for TP1 to TP3 and BATCH). Gupta and Ravindran extend branch–and–bound to the discrete case by defining branches similar to integer branches in this case. In a separate study Gupta and Ravindran [36] investigate three different branching rules and heuristics for upper bounds and node selection. They conclude that the branching rule which branches on the most fractional variable first is the best and that the heuristics do not play a great role in reducing the CPU time. More recently Sandgren [64], [65] gives an implementation of a branch–and–bound routine. He prefers to monitor the CPU time required to solve each node in the tree and to abandon a node temporarily if it takes too much time. Hajela and Shih [37] also give an implementation of a branch–and–bound routine and apply

it to the optimal sizing of laminated composites.

Körner [45] proposes a branching heuristic that aims to choose the branching variables in an order which minimized the size of the tree. His rule is a generalization of an earlier rule for integer quadratic programs [44]. It uses second order information to estimate the size of the tree that will result by branching on certain variables first. He shows how the method can be implemented efficiently and carries out numerical experiments. He concludes that the number of nodes in the branch–and–bound tree is an order of magnitude smaller with his branching rule than without it.

Another important deterministic method is Benders Decomposition, which was first introduced by Benders [6] for problems of the form

$$\begin{cases} \min\limits_{x,y} & c^T x + f(y) \\ \text{subject to} & A^T x + g(y) \leq 0 \\ & x \in X,\ y \in Y \end{cases}$$

where y is the vector of so called *complicating variables*. Benders gives two examples for complicating variables where the y_i are integer or occur nonlinearly in the problem. First the problem is decomposed into a minimization over y and a minimization over x of an LP problem parameterized in y. The LP is dualized and an equivalent master program in y only is obtained. Unfortunately, this master program is defined implicitly in terms of optimal LP solutions for fixed y, so that it cannot be solved directly. Instead a relaxation based strategy is proposed in which a finite number of LP–subproblems and master program relaxations that include the complicating variables are solved. Employing nonlinear duality theory Geoffrion [26] generalizes Benders' decomposition to problems like P, where f and g are convex in x for fixed y and in addition a so called *Property* $\mathcal{P}$, which is described in Section 4.3, holds.

Flippo, Rinnoy Kan and van der Hoek [20] show in their elegant discussion of decomposition techniques that Geoffrion's Property $\mathcal{P}$ is implied by the convexity of f and g. They derive the master program in a more general setting and obtain very elegant convergence results. The main result is that Benders Decomposition converges whenever either the primal feasible set Y is finite (as for instance in MINLP) or the dual feasible set is finite. An important example where the dual can be solved by enumerating a finite number of vertices of the dual space is given by Chandra and Dixon [12]. Chandra and Dixon apply Benders' Decomposition to the constrained l_1 problem

$$\begin{cases} \min\limits_{x} & \|b - Ax\|_1 \\ \text{subject to} & Cx \leq d. \end{cases}$$

Since the l_1 norm is a polyhedral norm, the dual space has a finite number of vertices (e.g. [17]) and an appropriate finite convergence result follows immediately. This last argument also indicates that a similar result holds for the constrained l_∞ problem.

Practical implementation issues of Benders' Decomposition are discussed by Sahinidis and Grossmann [62]. They also show, how Benders' cuts can be interpreted as outer approximations of the value function

$$v(y) = \begin{cases} \min\limits_{x} & f(x,y) \\ \text{subject to} & g(x,y) \leq 0 \\ & x \in X \end{cases}$$

defined, for instance, by Geoffrion [25].

Lazimy [48], [49] applies Benders' Decomposition to a class of MIQP problems. His approach is rather complicated and he makes an unnecessary assumption on the rank of the Hessian matrix, thus not recovering MILP problems as a special case. Flippo and Rinnoy Kan [21] show that Lazimy's approach is based on an inaccurate interpretation of Benders' Decomposition and they give a correct Benders' Decomposition for the MIQP problem.

A very similar approach to Benders' Decomposition is Outer Approximation which was introduced by Duran and Grossmann [15], [16] for a subclass of P whose objective and constraint functions are linear in the integer variables y. Instead of seeking a dual representation of the NLP–subproblems, first order necessary conditions are employed. The proposed algorithm solves a finite sequence of alternately NLP–subproblems (in which the integer variables are fixed) and MILP master programs. The optimal solution of the subproblem provides a point at which supporting hyperplanes of the functions are generated. These linearizations are collected in an MILP master program. The master program in turn determines a new integer assignment. Duran and Grossmann show that Outer Approximation provides stronger lower bounds than Benders' Decomposition and in a computational study [15] they conclude it to be superior to both Benders' Decomposition and branch–and–bound. Yuan, Zhang, Pibouleau and Domenech [79] generalize Outer Approximation to the wider class of problems represented by P.

The interpretation of Benders' Decomposition as an outer approximation procedure (see [62] and above) indicates that the two algorithms are very similar. In fact Yuan et.al. refer to their outer approximation algorithm erroneously as Benders' Decomposition. Outer Approximation can conversely be interpreted as Benders' Decomposition without the dualization of the NLP–subproblems. However, this dualization is really at the heart of Benders' Decomposition in that it ensures that the master program has a very special structure (with only a single continuous variable).

Outer Approximation has been implemented as a discrete optimizer DICOPT [43]. A severe limitation of Outer Approximation that was discovered early on is its failure to handle nonconvex problems appropriately. Consequently, heuristics were introduced to allow a more successful treatment of nonconvex functions (e.g. [42], [73]) and these are discussed in Chapter 8.

An alternative solution procedure, based on the same master program as Outer Approximation, is proposed by Quesada and Grossmann [60]. Instead of solving a sequence of NLP and MILP problems, the master program is updated during the MILP tree search whenever an integer feasible solution is encountered. This avoids the solution of related master programs and is advantageous in cases where the MILP master program is the bottleneck of the computation. Quesada and Grossmann give computational experience showing that this new approach can be superior to Outer Approximation. Two alternative versions of the algorithm are given in which convex combinations of the linearizations of the constraints are added to the master program, resulting in a master program with fewer constraints at the expense of a weaker MILP formulation.

Lagrangean Decomposition was proposed by Michelon and Maculan [54] for problems of the form

$$\begin{cases} \min\limits_{y} & f(y) \\ \text{subject to} & A^T y \leq b \\ & y \in Y \text{ integer} \end{cases}$$

extending results given by Guignard and Kim [34] who applied Lagrangean Decomposition to MILP problems. Lagrangean Decomposition defines a dual problem to the MINLP problem above. However, since some of the y_i are integer there exists usually a duality gap. Michelon and Maculan propose two algorithms based on Lagrangean Decomposition that aim at reducing the duality gap. Convergence results are given. Their second algorithm employs linear supporting hyperplanes on f in order to cut off any integer assignments that have already been visited by the algorithm. This makes it possible to interpret the master program in the Lagrangean Decomposition as an Outer Approximation master program. The difference lies in their objective function which is derived through subgradient techniques (see e.g. [56]). This observation makes an extension of their algorithm to problems like P very easy in view of the outer approximation scheme that is presented in Chapters 4 and 5.

There also exists a number of algorithms to solve pure 0-1 integer nonlinear problems (see e.g.

Balas and Mazzola [3] and [4]) in which the nonlinear problem is first reformulated as a multilinear problem and then this new problem is interpreted as a generalized covering problem for which cuts are derived. Balas and Mazzola give strong cuts and derive a class of algorithms.

1.4 Heuristic Methods in MINLP

If the amount of CPU time needed to produce an optimal solution with the algorithms of the previous section becomes too large, so that it becomes prohibitive to use a deterministic approach, then one has to resort to heuristic methods. These methods cannot give a guarantee that a solution is found but are usually faster than deterministic methods since they avoid the enumeration of many of the integer assignments. Most heuristic methods try to emulate successful continuous methods.

The need to solve practical MINLP problems has led to a large number of heuristics that have been developed over the years. These heuristics include a rounding procedure by Olsen and Vanderplaats [57], an attempted generalization of an SQP method by Cha and Mayne [10], [11], discrete gradient methods by Amir and Hasegawa [1], Bremicker, Loh and Papalambros [8] and Loh and Papalambros [52], [53], penalty function methods by Davydov and Sigal [14] and Han-Lin Li [50] and adaptive random search (e.g. Salcedo [63]).

The success of a heuristic method depends on its speed and on its reliability to locate good suboptimal solutions (in the sense that their objective value is not far from the optimum). Practitioners are often satisfied with a suboptimal solution provided the value of the objective function has been improved sufficiently. A lower bound on the optimal value is usually available from the solution of the NLP relaxation and this together with the values encountered at other integer assignments can provide a fairly good idea of the quality of a suboptimal solution.

The simplest heuristics are based on rounding. The MINLP problem is solved as an ordinary NLP problem by relaxing all integer restrictions and the solution is then rounded to a nearby integer point. The limitations of such an approach are clear: no guarantee can be given that the resulting point is even feasible and, moreover, its objective value might be far from the optimal objective value. An "intelligent" rounding procedure was developed by Olsen and Vanderplaats [57], who proposed a sequential discrete linearization scheme. The nonlinear functions f and g are approximated by linearizations about an initial point (x^0, y^0) and an MILP with a reduced set of variables is solved for a new iterate. The reduced set is obtained by restricting each integer variable to the values $y_i^0 - 1,\ y_i^0,\ y_i^0 + 1$. Olsen and Vanderplaats interpret this as an intelligent rounding procedure that attempts to find the best rounding combination based upon first order information.

Methods based on linearizations have proved to be very popular. Amir and Hasegawa [1], for example, generalize the steepest descent method with line search to MINLP problems. They reformulate P through the use of an inverse barrier function to obtain the following unconstrained MINLP

$$\min_{x \in X, y \in Y} \ \Phi(x, y) = f(x, y) + r \sum_{i=1}^{m} \frac{1}{g_i(x, y)}$$

where r is a monotonically decreasing sequence of positive numbers. The proposed method treats all continuous variables as discrete by introducing a resolution which depends on the size and range of the continuous variables. A generalization of the steepest descent vector is computed by scaling and rounding the continuous steepest descent vector. Next a discrete line–search is carried out and if this fails to improve the objective function a "subsequential search interval" is computed that includes neighbouring points along the search direction in the line–search. If this also fails, then the authors suggest to modify a set of orthogonal search directions in order to enable the procedure to follow curved valleys, similar to Rosenbrock's Orthogonalization Procedure. Unfortunately, Amir and Hasegawa do not address the important issue of increasing ill conditioning of the barrier

function and no convergence result is presented. Furthermore, experience with barrier methods in continuous optimization usually requires the solution of a sequence of barrier problems before their solution converges to the constrained minimizer. It seems to me that solving an MINLP problem by a sequence of unconstrained MINLP problems is unlikely to be effective.

Another method based upon linearizations is introduced by Bremicker, Paplambros and Loh [8] and Loh and Papalambros [52] and [53]. Their algorithm solves a finite sequence of MILP and NLP problems. Initially the integer variables are fixed and the resulting NLP–subproblem is solved. The nonlinear functions are then linearized about the optimal solution of the subproblem and an MILP problem is formed from these linearizations. The solution of the MILP provides a new integer point and the process is repeated. The algorithm is very similar to the outer approximation routine of Duran and Grossmann [15]. The main difference to outer approximation is that the linearizations are not collected in an MILP master program. Instead only the most recent linearizations are included in the MILP problem. Consequently, finite termination relies on heuristics and no guarantee can be given that the algorithm terminates at an optimal solution. Finite termination is forced by including a trust region in the MILP problem and the termination criterion used is that the difference between the incumbent and the current iterate is less than a given tolerance. Loh and Papalambros show that their algorithm is a descent algorithm and that it terminates finitely. They also claim that it is better suited to nonconvex MINLP problems, since linearizations that might cut into the feasible region are not collected in a master program. A suite of test problems, including several nonconvex MINLP problems, is solved to support this claim.

Sequential quadratic programming techniques for MINLP problems are not quite as popular as the sequential linear programming techniques described above. This is quite surprising considering the success of SQP methods in nonlinear programming, but might be explained by the lack of general purpose MIQP solvers. Cha and Mayne [10], [11] propose a "recursive QP" method for MINLP problems. Their algorithm obtains a search direction from a quadratic model. They prefer to use the SR1 formula to update the second order information instead of the BFGS or DFP formula (e.g. [17]). The authors quote poor experience with the DFP formula in discrete optimization for which they blame the sensitivity of the DFP formula to the accuracy of the line search, which is usually low in discrete optimization. A line–search is performed to obtain a new iterate. Once a good approximation to the minimum has been obtained, the discrete neighbourhood is searched by varying "two variables at a time".

Another class of heuristic algorithms is based upon penalty function methods. Among the earliest such method is the one proposed by Davydov and Sigal [14]. They observed that a binary variable y_i can be replaced by a continues unbounded variable, provided the constraint

$$y_i^2(1-y_i)^2 = 0$$

is added to the problem. They also use the representation of a general integer variable ($l_i \leq y_i \leq u_i$) based on the same idea of replacing the integer variable by a continuous unbounded variable and a constraint $\psi_i(y_i) = 0$, where

$$\psi_i(y_i) = \begin{cases} (y_i - l_i)^2 & y_i < l_i \\ (y_i - u_i)^2 & y_i > u_i \\ \sin^2(\pi y_i) & l_i \leq y_i \leq u_i \end{cases}$$

The authors then proceed to construct a penalty function and propose a scheme for solving the resulting problem. Although these reformulations are very attractive at first sight, they have a number of serious drawbacks. By introducing a nonlinear equality constraint, they transform a convex MINLP into a nonconvex NLP which is not easier to solve than the original MINLP and while there are rigorous methods for convex MINLP, the same is not true for general nonconvex

NLP. Moreover, most penalty functions require estimates of the Lagrange multipliers which are not known a priori and it might therefore become necessary to solve a sequence of nonconvex NLP problems. However, if a global NLP solver became available, then the methods based upon penalty function formulations would become deterministic methods.

Li [50] derives another penalty function approach by noting the following equivalence

$$y \in \{0,1\}^p \;\Leftrightarrow\; \begin{cases} 0 \le y_i \le 1,\ i = 1,\ldots,p \\ \\ \sum_{i=1}^{p} (y_i - y_i^2) = 0 \end{cases}$$

He thus transforms the MINLP into an NLP by replacing general integer variables by their binary representation and by replacing the integer restrictions by the nonlinear constraint above. The resulting NLP is clearly nonconvex. To resolve the NLP, Li suggests to use a Courant penalty function (e.g. [17]) so that the NLP

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{subject to} & c_i(x) = 0,\ i = 1,\ldots,m \\ & x \in X \end{cases}$$

becomes the unconstrained problem

$$\min_{x \in X}\ \Psi(x,\sigma) = f(x) + \tfrac{1}{2}\sigma \sum_{i=1}^{m} (c_i(x))^2$$

In order to avoid the ill–conditioning associated with increasing σ Li proposes to solve instead the following problem

$$\begin{cases} \min\limits_{x \in X} & \Psi(x,\sigma) \\ \\ \text{subject to} & \tfrac{1}{2}\sum_{i=1}^{m} (c_i(x))^2 \le \sigma \end{cases}$$

The proposed algorithm solves a sequence of the above problem for increasing penalty parameters σ until the difference between two successive iterates becomes sufficiently small and the iterates approach a 0-1 vector. If the algorithm converges to a solution which is not a 0-1 vector then the procedure is restarted from an alternative initial point. Li's algorithm suffers from the same drawbacks as Davydov and Sigal's approach and he does not show how these can be overcome.

Another class of popular methods both for global optimization and MINLP problems are stochastic methods. An example of such an algorithm is Salcedo's Adaptive Random Search [63]. The method generates a set of random test points that are continuous and/or integer. The continuous test points that are generated are centered at their current optimum, while the integer test points are simply feasible (i.e. they lie in Y). The algorithm terminates after a given number of iterations in which the search regions for the continuous variables is contracted.

Heuristics could play an important part in solving MINLP problems if all deterministic methods fail. Alternatively, it is possible to truncate deterministic methods by aborting the tree search after a given amount of time. In this case the incumbent provides a suboptimal solution.

1.5 Main Assumptions

This thesis focuses on solving MINLP problems by deterministic methods which require a number of assumptions on the structure of the model problem P. These assumptions are considerably weaker

than the assumptions used by Duran and Grossmann [15], thus enlarging the class of problems that can be treated by outer–approximation. Some of these assumptions can be relaxed further and it is indicated how this can be achieved.

The following assumptions are made

A1 X is a nonempty compact convex set defined by a system of linear inequality constraints and the functions

$$\begin{aligned} f : \mathbb{R}^n \times \mathbb{R}^p &\to \mathbb{R} \\ g : \mathbb{R}^n \times \mathbb{R}^p &\to \mathbb{R}^m \end{aligned}$$

are convex.

A2 f and g are once continuously differentiable.

A3 A constraint qualification holds at the solution of every NLP subproblem which is obtained from P by fixing the integer variables y. This could be, for instance, the assumption that the set of feasible directions at the solution can be identified with the set of feasible directions for the constraint linearized at the solution (e. g. [17], p. 202), for which a sufficient condition is that the normal vectors of the active constraints are linearly independent.

A4 The NLP subproblems obtained by fixing the integer variables in P can be solved exactly.

The most serious restriction on P is the convexity assumption **A1**. This assumption is necessary since it ensures that the linearizations of the objective and constraint functions are indeed outer approximations. Although any deterministic algorithm can be applied to a nonconvex problem P, no guarantee can be given in that case that an optimal solution will be found. Under additional assumptions on the structure of f it is shown in Chapter 8 that the convexity assumption can be relaxed if proper linear underestimators can be found. The framework presented in Chapter 8 also allows assumption **A4** to be dropped through the use of suitable tolerances.

The differentiability assumption **A2** can be relaxed by replacing the gradients by subdifferentials. In Chapter 7 outer approximation is generalized to a class of nonsmooth MINLP problems which are almost everywhere continuously differentiable.

In order to find a stationary point most NLP solvers solve the Kuhn-Tucker conditions which require a constraint qualification to hold. Since all deterministic methods solve NLP problems at some stage, assumption **A3** is needed to ensure that these NLP problems are solved correctly. However, for outer approximation the constraint qualification is more important than has previously been recognised. It is commonly regarded as a technical assumption [15] that ensures that first order necessary conditions hold at the solution of every NLP subproblem. It is important to realize that if **A3** did not hold, then an integer assignment might be generated again by the algorithms of Chapter 5. An example where the absence of a regularity assumption leads to cycling in outer approximation is provided in Chapter 5.

1.6 Synopsis of the Thesis

A brief review of some of the important features in Nonlinear Programming is presented in Chapter 2. The concepts of convexity, Lagrange multipliers, duality and optimality conditions are introduced and methods for solving NLP problems, like SQP and the Simplex Method are presented. It is assumed throughout this thesis that good NLP, QP and LP algorithms exist. The concepts and methods of Chapter 2 provide a foundation for the use of NLP in subsequent chapters.

The main concern of this thesis is deterministic methods for MINLP problems and a number of such methods are studied. Chapter 3 presents a general framework for nonlinear branch–and–bound. This can itself be a very successful MINLP solver and underpins all the other methods of

the following chapters by providing a solver for the different master programs. The introduction of MIQP master programs in Chapter 5 caused me to become interested in MIQP problems and a detailed study of branch–and–bound for MIQP problems is given. It is shown, that it is possible to derive new improved lower bounds based on taking a step of the dual active set method. It is also explained how the lower bounds can be computed efficiently using the Linear Complementarity formulation for QP problems and numerical experience with an MIQP branch–and–bound algorithm is presented.

The study of MINLP solvers concentrates on an outer approximation scheme proposed by Duran and Grossmann [15]. In this scheme, problem P is reformulated as an equivalent MILP master program, relaxations of which are used in an iterative scheme. The reformulation is presented in Chapter 4 and the algorithms based upon it are studied in Chapter 5. The reformulation of the model problem P, presented in Chapter 4 improves on two earlier reformulations by Duran and Grossmann [15] and Yuan et.al. [79]. New insight is gained into outer approximation. The new reformulation produces a master program which has potentially fewer constraints and also corrects an inaccuracy which occurs in [15] and [79]. The chapter finishes with the derivation of the Benders' Decomposition master program which follows largely the derivation by Flippo et.al. [20]. It allows a direct comparison of the two sets of cutting planes generated by outer approximation and Benders' Decomposition, which shows the former to generate the stronger cuts.

The MILP master program developed in Chapter 4 cannot be solved directly, since it is only defined implicitly and requires the solution of an NLP (in which the integer variables are fixed) for each set of supporting hyperplanes that is generated. This suggests the use of an iterative scheme and an algorithm is presented in Chapter 5 which is shown to iterate finitely between the solution of NLP subproblems and MILP master program relaxations. This generalizes the original outer approximation algorithm by Duran and Grossmann [15] to a more general class of problems. An alternative scheme, due to Quesada and Grossmann [60], in which the master program is updated during the MILP branch–and–bound tree search, is also generalized. Based upon the MILP master program a generalization of Lagrangean Decomposition is given.

The worst case behaviour of the two outer approximation algorithms is examined and an example is given for which both algorithms visit all integer assignments, while the same example is solved after only one branch by a branch–and–bound routine. The worst case behaviour is explained by the inability of the algorithms based on outer approximations to take curvature information into account. Two new algorithms are proposed that include a curvature term into the master program relaxations. Implementation issues are discussed for the algorithms.

Numerical experience with the outer approximation algorithms and an implementation of an MINLP branch–and-bound solver is presented in Chapter 6. This shows that no one solver outperforms all other solvers on all problems. However, the test problems can be grouped into classes of problems and a "rule of thumb" is motivated which indicates which solver is likely to perform best for each class of problem.

The application of outer approximation to nonsmooth MINLP problems is discussed in Chapter 7. Apart from being of interest in their own right, nonsmooth MINLP problems arise whenever penalty function formulations of MINLP problems are solved. The results of the previous chapters are generalized by replacing the gradient in the analysis by the subdifferential and applying suitable optimality conditions. A more convenient interpretation of these conditions in the case of the l_1 exact penalty function is also given.

The most serious restriction on the functions f and g in the model problem P is that all functions are required to be convex. The reason for this convexity assumption is that otherwise the linearizations of f and g are not supporting hyperplanes and the outer approximation algorithms might terminate without finding an optimal solution. Chapter 8 is an attempt to address these difficulties. A general framework is presented for the solution of a class of nonconvex MINLP prob-

lems. Besides giving rise to rigorous methods for some nonconvex MINLP problems the framework also allows the creation of heuristic methods based upon linear underestimators and tolerances. The framework also removes the necessity for assumption **A4** for an exact NLP solver through the use of suitable tolerances.

Chapter 2

Concepts in Nonlinear Programming

2.1 Introduction

This chapter reviews briefly some important aspect of Nonlinear Programming (NLP) problems, which are used in the later chapters. The first section gives a brief introduction to the notation and terminology used in nonlinear programming. In Section 2.2 optimality conditions for the model NLP problem *NLP*, defined below, are derived and the Lagrange multipliers and the Lagrangian function are introduced. The concepts of convexity and duality are presented in Section 2.3. Methods for NLP problems are discussed in Section 2.4. The chapter closes with a review of exact penalty functions and optimality conditions for nonsmooth NLP problems in Section 2.5.

The problem that is considered here is

$$NLP \begin{cases} \min\limits_{x} & f(x) \\ \text{subject to} & g_i(x) = 0\ ,\ i \in E \\ & g_i(x) \geq 0\ ,\ i \in I, \end{cases}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, the *objective function*, and $g_i : \mathbb{R}^n \to \mathbb{R}$, the *constraint functions*, are continuously differentiable and the finite sets E and I index the equality and inequality constraints respectively. Any of the two sets E and I may be empty and if both E and I are empty then *NLP* is an unconstrained optimization problem. The set of all feasible points is denoted by

$$R = \{x \mid g_i(x) = 0,\ i \in E,\ \text{ and } g_i(x) \geq 0,\ i \in I\}$$

and is called the *feasible region*.

A point x' which satisfies all constraints is called a *feasible point* and a feasible point x^* is called a *local minimizer*, if and only if

$$f(x^*) \leq f(x)\ ,\ \forall x \in N(x^*) \cap R$$

where $N(x^*)$ is some neighbourhood of x^*. The point x^* is called a global minimizer, if and only if

$$f(x^*) \leq f(x)\ ,\ \forall x \in R$$

The problem of finding a global minimum is very hard and usually intractable except if *NLP* is a convex programming problem (c.f. Section 2.3). In the remainder of this chapter only local minimizers are therefore considered.

The set of indices $i \in I \cup E$ whose corresponding constraints hold with equality at a feasible point x' is called the *active set* and denoted by

$$\mathcal{A}' = \{i \mid i \in E \cup I \text{ and } g_i(x') = 0\}.$$

Clearly $E \subset \mathcal{A}$ for any feasible point.

Problem *NLP* includes a number of important sub–classes of problems. If both f and g_i are linear functions, then *NLP* is a Linear Programming (LP) problem. If f is a quadratic function and g_i are linear, then *NLP* is referred to as a Quadratic Programming (QP) problem.

2.2 Lagrange function, multipliers and optimality conditions

It is possible to derive stationary point conditions for *NLP*, similar to the stationary point conditions in unconstrained optimization ($\nabla f = 0$). These conditions introduce the notion of Lagrange multipliers and can be seen to be equivalent to a certain stationary point condition for a Lagrangian function which is also introduced.

In deriving optimality conditions it is important to realize that only the active constraints can play a part in them, since for any inactive constraint $g_i(x^*) > 0$ there exists a neighbourhood $N(x^*)$ such that $g_i(x) > 0 \; \forall x \in N(x^*)$. Let x^* be a local minimum and let $\mathcal{A}^*$ denote the corresponding active set. Considering the Taylor expansion of g_i about x^*, where δ is a feasible incremental step, gives

$$\begin{aligned} g_i(x^* + \delta) &= g_i^* + \delta^T a_i^* + o(\|\delta\|) \\ &= 0 + \delta^T a_i^* + o(\|\delta\|) \quad \forall i \in \mathcal{A}^*, \end{aligned}$$

where $a_i^* = \nabla g_i^*$. Since $g_i(x^* + \delta) = 0$ for $i \in E$ and positive for $i \in \mathcal{A} \cap I$ it is necessary for δ to lie along the line s that satisfies

$$\begin{aligned} s^T a_i^* &\geq 0 \quad \forall i \in \mathcal{A}^* \cap I \\ s^T a_i^* &= 0 \quad \forall i \in E. \end{aligned}$$

Any s that satisfies these conditions is called a *feasible direction*. If a regularity assumption holds, that is if it is possible to find an incremental step δ along a feasible direction s, then the following lemma states an optimality condition for a local minimizer.

Lemma 2.2.1 *Let*

$$\begin{aligned} F^* := \{ s \mid \; & s^T a_i^* = 0 \; , \; \forall i \in E \\ & s^T a_i^* \geq 0 \; , \; \forall i \in \mathcal{A}^* \cap I \\ & s^T \nabla f^* < 0 \} \end{aligned}$$

be the set of all feasible descent directions. Then $F^* = \emptyset$ if and only if there exist Lagrange multipliers $\lambda_i^*, \; i \in \mathcal{A}^*$ such that

$$\begin{aligned} \nabla f^* &= \sum_{i \in \mathcal{A}^*} \lambda_i^* \nabla g_i^* \\ \lambda_i^* &\geq 0 \; , \; \forall i \in \mathcal{A}^* \cap I \end{aligned}$$

The proof of Lemma 2.2.1 is a famous result due to Farkas and can be found for example in ([17], p. 199). A condition which implies the regularity assumption is called a constraint qualification. A simple constraint qualification is that all $\{a_i^*, \; i \in \mathcal{A}^*\}$ are linearly independent (e.g. Assumption **A3**).

Lemma 2.2.1 gives a first order condition which can be readily checked and which implies that no linearized feasible descent direction exists, if and only if the gradient of the objective function is a linear combination of the gradients of the active constraints where the weights are given by the respective Lagrange multipliers.

Before stating the optimality conditions for *NLP*, the *Lagrangian function*, a weighted linear combination of the objective function and the constraint functions, is defined as

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in I} \lambda_i g_i(x) - \sum_{i \in E} \lambda_i g_i(x)$$

Now the Kuhn–Tucker first order necessary conditions for *NLP* can be stated.

Theorem 2.2.1 *If x^* is a local minimizer of* NLP *and if a regularity assumption holds at x^*, then there exist Lagrange multipliers λ^* such that*

$$\nabla f^* - \sum_{i \in \mathcal{A}^*} \lambda_i^* \nabla g_i^* = 0 \tag{2.1}$$

$$g_i^* = 0 \ , \ \forall i \in E \tag{2.2}$$

$$g_i^* \geq 0 \ , \ \forall i \in I \tag{2.3}$$

$$\lambda_i^* \geq 0 \ , \ \forall i \in I \tag{2.4}$$

$$\lambda_i^* g_i^* = 0 \ , \ \forall i \tag{2.5}$$

Condition (2.1) is often referred to as first order condition, (2.2) and (2.3) are primal feasibility, (2.4) is dual feasibility and (2.5) is referred to as the complementarity condition.

Lagrange multipliers not only play an important role in the optimality conditions, but also give the rate of change of the solution, if the constraints g_i are perturbed. This is often exploited in post–optimality or sensitivity analysis. If $c_i(x) = 0$ is perturbed to $c_i(x) = \epsilon_i$, then setting $x = x(\epsilon)$, as the solution of the perturbed problem, it follows that $f(x(\epsilon)) = \mathcal{L}(x(\epsilon), \lambda(\epsilon), \epsilon)$ and using the chain rule it is possible to estimate the first order change in f upon a small perturbation in the constraints as

$$\frac{df}{d\epsilon_i} = \frac{d\mathcal{L}}{d\epsilon_i} = \frac{\partial x^T}{\partial \epsilon_i} \nabla_x \mathcal{L} + \frac{\partial \lambda^T}{\partial \epsilon_i} \nabla_\lambda \mathcal{L} + \frac{\partial \mathcal{L}}{\partial \epsilon_i} = \lambda_i$$

since the optimality of x implies that $\nabla_x \mathcal{L} = 0$ and $\nabla_\lambda \mathcal{L} = 0$.

Any point x^* that satisfies the Kuhn–Tucker condition of Theorem 2.2.1 is called a *Kuhn–Tucker* or *stationary point*. In order to determine the nature of a stationary point it becomes necessary to consider additional conditions, since the Kuhn–Tucker conditions of Theorem 2.2.1 are only necessary conditions. Similar to unconstrained optimization, it is possible to derive second order sufficient conditions for a local minimum of *NLP*.

Theorem 2.2.2 *If x^* is a Kuhn-Tucker point, if strict complementarity holds (i.e. $\lambda_i^* > 0, \ \forall i \in \mathcal{A}^* \cap I$) and if*

$$s^T \nabla^2 \mathcal{L}(x^*, \lambda^*) s > 0 \ , \ \forall s : s^T a_i^* = 0, \ i \in \mathcal{A}^*$$

then x^* is a strict local minimizer of NLP.

Unless additional assumptions on *NLP* are made there exists a gap between necessary and sufficient conditions. An assumption which removes this gap is given in the next section.

2.3 Convexity and Duality

The Kuhn–Tucker conditions of the Section 2.2 are only necessary conditions and as such they do not provide a characterization of optimal points. The concept of convexity together with a regularity assumption ensures that the Kuhn–Tucker conditions become necessary and sufficient conditions for a global minimum. Moreover, convex problems also possess a related problem called the dual whose solution is under certain conditions equivalent to the solution of *NLP*. The convexity assumption of Chapter 4 turns out to be the crucial condition that ensures that the proposed outer approximation reformulation is correct. Unfortunately, many important practical problems are not convex.

A set $C \subset \mathbb{R}^n$ is called *convex*, if and only if for any two points in C the line connecting these two points also lies in C, that is if and only if

$$x^0, x^1 \in C \;\Rightarrow\; x^\theta := (1-\theta)x^0 + \theta x^1 \in C \;,\; \forall \theta \in [0,1].$$

Examples of convex sets are half spaces, spheres and the feasible region of an LP or a QP problem.

A function f is called *convex* if and only if its epigraph is a convex set, that is if and only if

$$f^\theta \leq (1-\theta)f^0 + \theta f^1 \;,\; \forall \theta \in [0,1]$$

where $f^j = f(x^j)$. If f is continuously differentiable this condition can be seen to be equivalent to the condition that any tangent plane on f is a supporting hyperplane, that is

$$f^1 \geq f^0 + (\nabla f^o)^T (x^1 - x^0).$$

For a twice continuously differentiable function, convexity is equivalent to the positive semi–definiteness of the Hessian matrix $(\nabla^2 f)$. Linear functions and norms are examples of convex functions. A quadratic $f(x) = \frac{1}{2}x^T G x + g^T x + c$ is convex, if its Hessian G is positive semi–definite and it is strictly convex if G is positive definite.

A convex programming problem is a problem with a convex objective function and a convex feasible region R. A sufficient condition for R to be convex is that $g_i,\ i \in E$ are linear and that $g_i,\ i \in I$ are concave (i.e. $-g_i$ is convex). Convex programming problems (CPP) possess a number of attractive features. For instance, any local solution to a CPP is a global solution and the set of global solutions is convex. If f is strictly convex, then the global minimum is unique, provided that it exists. More important though, the Kuhn–Tucker conditions become sufficient conditions for a global minimum of a CPP.

An important property of a CPP is that it possesses a so called *dual* problem whose solution is equivalent to the solution of the original or *primal* problem in the sense indicated below. If *NLP* is a CPP and if $\mathcal{L}$ is differentiable with respect to x, then the *Wolfe dual* is defined as

$$D_W \begin{cases} \max\limits_{x,\lambda} & \mathcal{L}(x,\lambda) \\ \text{subject to} & \nabla_x \mathcal{L}(x,\lambda) = 0 \\ & \lambda_i \geq 0 \;,\; i \in I \end{cases}$$

Any feasible point of the Wolfe dual provides a lower bound on the primal problem *NLP* and this is known as *weak duality*. Moreover, under a regularity assumption, if x^* solves *NLP*, then x^*, λ^* solves D_W and the function values are equal which is referred to as *strong duality*. The convexity of *NLP* is essential in the proof for weak and strong duality for the Wolfe dual.

An alternative dual formulation is given by the *Lagrangian dual*. This dual formulation is used in Chapter 4 to derive Benders Decomposition. If the primal problem is written as

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{subject to} & g(x) \geq 0 \\ & x \in X \end{cases}$$

then the dual is defined as

$$D_L \left\{ \max_{\lambda \geq 0} \left(\inf_{x \in X} \mathcal{L}(x,\lambda) \right) \right.$$

Again it is possible to derive weak and strong duality results as for the Wolfe dual. However, unlike for the Wolfe dual, weak duality does *not* require a convexity assumption and this is seen from the following manipulations. For $\lambda' \geq 0$, it follows that if x' is feasible in *NLP*, then

$$\inf_x \mathcal{L}(x,\lambda') \leq \mathcal{L}(x',\lambda') \leq f(x')$$

and weak duality follows.

Both duals are related to one another and Geoffrion [25] remarks that if $\lambda' \geq 0$ is fixed, then it follows from the first order conditions that (x', λ') is feasible in D_W if and only if x' is the unconstrained minimizer of $\mathcal{L}(x, \lambda')$. Hence it is possible to write D_W as

$$D'_W \begin{cases} \max\limits_{\lambda \geq 0} & \min\limits_{x} \mathcal{L}(x, \lambda) \\ \text{subject to} & \lambda : \exists x \text{ which achieves the unconstrained minimum of } \mathcal{L}(x, \lambda) \end{cases}$$

and apart from the additional constraint on λ, this is identical to D_L and Geoffrion [25] recovers all duality results for the Wolfe dual from his results for the Lagrangian dual.

If *NLP* is not convex, or if *NLP* is an MINLP, then the Wolfe dual is not defined and the Lagrangian dual usually exhibits a duality gap, that is the optimal value of the dual is strictly less than the optimal value of the primal. However, weak duality still holds for D_L and this is exploited in the derivation of Lagrangian Decomposition in Chapter 5, which gives rise to an MINLP algorithm which iteratively reduces the duality gap until convergence occurs.

2.4 Methods

There exist many methods for solving NLP problems, both in their general form and for particular sub–classes of problems such as LP or QP problems. The methods that underlie this study fall into the class of *Active Set Methods* (ASMs). A common feature of an ASM is that with every iterate x^k there is associated an active set $\mathcal{A}^{(k)}$ of constraints. The constraints in $\mathcal{A}^{(k)}$ are temporarily assumed to hold as equalities. A new iterate x^{k+1} is computed and the active set is updated.

If both the constraints and the objective function are linear, then the problem is an LP problem and the ASM can take advantage of this special structure. The feasible region of an LP is a polytope and there exists a solution which lies at a vertex. Hence the ASM moves from one vertex to another updating the active set and reducing the objective function. At a vertex, multipliers are computed and if all multipliers are non–negative then the vertex is optimal. Otherwise there exists a negative multiplier, λ_q say, and the objective function can be reduced by moving away from constraint q. If A denotes the matrix whose columns are the constraint normals of the active constraints, then a direction that moves away from constraint q and along which the objective is reduced is given by the row of A^{-1} corresponding to q. A ratio test determines which inactive constraint becomes active and the first such constraint, p say, is exchanged with q in the active set and the whole process is repeated.

The ASM for LP problems is finite, provided degeneracy does not occur. There exist algorithms that render the ASM finite if degeneracy occurs even in inexact arithmetic [18]. It is not difficult to show that the ASM is equivalent to the Simplex Method. In practice, the inverse of A is not computed directly, but instead LU factors or product form of A are used to operate with the inverse.

QP problems form another sub–class of NLP problems where it is possible to take advantage of the special structure of the problem. Since the objective function is no longer linear the solution does not necessarily lie at a vertex of the feasible region. Consequently, the optimal active set $\mathcal{A}^*$ might contain anything between 0 and n constraints. The active set defines an equality constrained QP (EP) and the primal ASM for QP problems moves from one feasible solution to an EP (FSEP) to another, reducing the objective. At an FSEP, multipliers of the active constraints are determined and if all multipliers are non–negative then this FSEP is an optimal solution to the original QP. Otherwise, a negative multiplier, say λ_q, is chosen and constraint q is dropped from the active set by moving towards the solution of the new EP. A ratio test determines the first inactive constraint that becomes newly active. Two cases can occur: either the ASM moves to the solution of the EP, in which case multipliers are computed and the optimality test is repeated, or a new constraint

becomes active, which is taken into the active set and a new EP is solved. The iterates of the primal ASM are feasible points which satisfy the first order condition (2.1), but – apart from the final iterate – not dual feasibility ($\lambda_i \geq 0,\ \forall i \in I$).

Another ASM for QP problems is the dual ASM, whose iterates satisfy the first order condition (2.1) and dual feasibility, but are primal infeasible. It requires the Hessian to be positive definite. At each iteration a negative residual, r_q say, is chosen and q is added to the active set. A move towards the solution of the new EP is made and if $a_q \in$ span $\{a_i,\ i \in \mathcal{A}\}$ then any index of an inequality constraint that becomes positive during the move is dropped from the active set $\mathcal{A}$. If any $\lambda_i \searrow 0$ for $i \in \mathcal{A} \cap I$ then the corresponding constraint is removed from the active set and a new EP is solved. The algorithm terminates as soon as primal feasibility is obtained.

The primal and the dual ASMs for QP terminate in a finite number of steps provided degeneracy does not occur. Fletcher [18] gives a primal ASM with a finite termination guarantee even in inexact arithmetic. The LP and QP solver that underlies this thesis is an implementation of a primal ASM with the additional feature of handling degeneracy.

Another class of methods that differs conceptually from ASMs are interior point methods. These methods possess the attractive feature of proven polynomiality, while ASMs can in theory exhibit exponential worst case behaviour. In practice, however, ASMs perform well and interior point methods only seem to become competitive for very large problems. Since the size of MINLP problems that is manageable is rather small, ASMs are used in this thesis. Moreover, integer programming algorithms frequently solve a sequence of closely related QP or LP problems and with an ASM full advantage can be taken of advanced basis and existing factorizations, so that ASM are very competitive in this area.

A very successful method for solving NLP problems is the Sequential Quadratic Programming (SQP) method. It is most easily explained for the equality constrained problem

$$\begin{cases} \min\limits_{x} & f(x) \\ \text{subject to} & c(x) = 0 \end{cases}$$

as applying Newton's method to the nonlinear equations ((2.1) and (2.2)) that arise from the KT conditions. The resulting linear system that is solved at every iteration of the Newton iteration can be interpreted as the KT conditions of a quadratic approximation to the NLP problem. This quadratic approximation is obtained by making a linear approximation to the constraints and a quadratic approximation to the objective augmented with second order information from the constraints. At each iteration of SQP a QP problem

$$\begin{cases} \min\limits_{x} & q(\delta) = \frac{1}{2}\delta^T W^k \delta + (\nabla f^k)^T \delta + f^k \\ \text{subject to} & l(\delta) = c^k + (\nabla c^k)^T \delta = 0 \end{cases}$$

is solved for a step δ and the new iterate is $x^{k+1} = x^k + \delta$. The Hessian W^k is a second order term of the Lagrangean, $\nabla^2 \mathcal{L}^k$. Under mild assumptions it can be shown that the SQP method converges locally at second order rate.

A number of practical considerations play an important part in SQP methods. The second order information that is needed in W^k is usually not available and various ways have been suggested to use quasi–Newton updates for W^k which only require first order information. Another important feature that many implementations possess is a way of forcing global convergence to a local minimum. This can be achieved by either performing a line–search using a suitable merit function (e.g. a penalty function presented in the next section) or by including a trust region in the QP. The NLP solver that underlies this work uses a quasi–Newton update of W^k and has an augmented Lagrangean merit function.

2.5 Penalty Functions and their Optimality Conditions

Exact penalty functions offer the possibility of solving NLP problems by one application of a solver for unconstrained problems. Unfortunately, exact penalty functions are non–smooth. This section introduces exact penalty functions and optimality conditions for more general non–smooth functions. The concepts introduced in this section are particularly relevant to Chapter 7.

If the functions in the NLP problem are not continuously differentiable, then the first order conditions are meaningless, since the gradients might not exist. In this section composite functions are introduced which can be seen to incorporate a large class of non–smooth functions. The absence of differentiability then motivates the introduction of the concepts of subgradient and subdifferential. Using these concepts it is possible to derive first and second order optimality conditions for a local minimizer under certain regularity assumptions. An important subclass of nonsmooth optimization (NSO) problems is formed by a certain class of exact penalty functions which are considered towards the end of this section.

A useful concept in the discussion of NSO problems are *composite functions* which can model a great variety of nonsmooth applications and are defined by

$$\Phi(x) = f(x) + h(g(x))$$

where f and g are as in the Section 2.2 (i.e. once continuously differentiable) and $h : \mathbb{R}^m \to \mathbb{R}$ is convex but nonsmooth (i.e. continuous). The model problem considered here can now be stated as

$$NSO \left\{ \begin{array}{ll} \min\limits_{x} & f(x) + h(g(x)) \\ \text{subject to} & t(r(x)) \leq 0, \end{array} \right.$$

where f, g, and r are once continuously differentiable and h and t are convex and nonsmooth. Examples for composite functions are

$$\begin{array}{l} h(g) := \max(g_i) \\ h(g) := \|g\| \quad \text{or} \quad h(g) := \|g^+\| \end{array}$$

where $a^+ := max(-a, 0)$ is defined componentwise for the vector g and $\|\cdot\|$ is any norm. The examples involving a norm are commonly used in exact penalty functions which are discussed towards the end of this section.

In order to derive first order necessary conditions it is convenient to introduce the concept of a subdifferential, which replaces the gradient in nonsmooth optimization. For a convex function f, defined on a convex set C, and a point x that lies in the interior of C, a supporting hyperplane at x satisfies

$$f(x + \delta) \geq f(x) + d^T\delta \ , \ \forall x + \delta \in C$$

where d is the normal vector of the supporting hyperplane. If f is continuously differentiable, then $d = \nabla f$. Such a vector d is called a *subgradient* and the *subdifferential* ∂f of f is the set of all subgradients, that is

$$\partial f(x) := \{d \mid f(x + \delta) \geq f(x) + d^T\delta \ , \ \forall x + \delta \in C\}.$$

Thus the subdifferential can be interpreted as the set of all supporting hyperplanes as illustrated in Figure 2.1.

Under a regularity assumption it is now possible to state the first order necessary conditions for a local minimizer of NSO. Such a regularity assumption is implied, for instance, by the condition that $t(r)$ is locally linear about x^* and that $r(x)$ are affine functions.

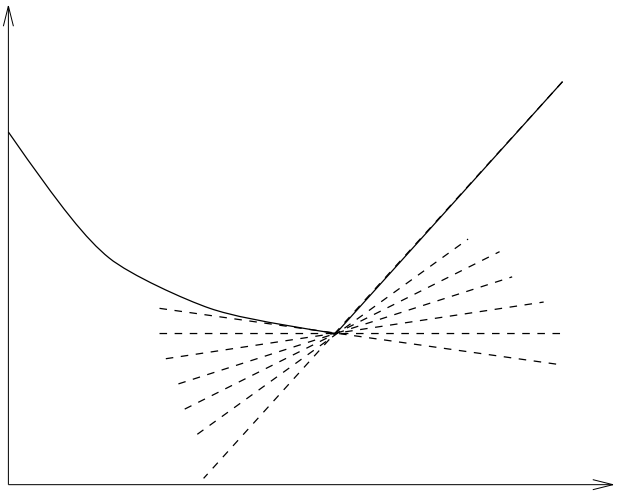

Figure 2.1: Outer approximation of a nonsmooth function by its subdifferential

Theorem 2.5.1 *If x^* solves locally NSO and a regularity assumption holds, then there exists multipliers $\lambda^* \in \partial h^*$ and $u^* \in \partial t^*$ and $\pi^* \geq 0$ such that*

$$\begin{aligned} 0 &= \nabla \mathcal{L}(x^*, \lambda^*, u^*, \pi^*) \\ &= d^* + A^*\lambda^* + \pi^* R^* u^* \\ t^* &\leq 0 \\ \pi^* t^* &= 0 \end{aligned}$$

where $A^* := [\nabla g^*]^T, R^* := [\nabla r^*]^T$ and $d^* := \nabla f^*$.

These conditions are very similar to the Kuhn–Tucker conditions of Section 2.2 and it is again possible to interpret them as stationary point conditions for a Lagrangian function defined by

$$\mathcal{L}(x, \lambda, u, \pi) := f(x) + \lambda^T g(x) + \pi u^T r(x).$$

An important class of NSO problems is given by *exact penalty functions*. These functions offer a framework by which it is possible to restate a constrained NLP problem as an unconstrained problem. The term penalty function is derived from the fact that a penalty term is added to the objective function that penalizes constraint violations. They are called exact, since – at least in theory – the constrained minimizer can be obtained in a single application of an unconstrained optimizer to the penalty function.

In the remainder of this section it is assumed that E is empty. An *exact penalty function* for *NLP* is given by

$$\Phi(x) = \nu f(x) + \|g(x)^+\|$$

where $\|\cdot\|$ is any norm and ν is the *penalty parameter* and the unconstrained problem that is solved instead of *NLP* is defined as

$$EPF \Big\{ \min_x \Phi(x)$$

Problem *EPF* is a convex programming problem, if f and g are convex.

It is not difficult to derive optimality conditions for *EPF* from Theorem 2.5.1 and this is done next. However, in order to obtain more convenient expressions for the subdifferential $\partial h(g)$, this is

simplified first. If $\|\cdot\|_D$ denotes the *dual norm* to $\|\cdot\|$ defined by

$$\|\lambda\|_D := \max_{\|g\|\leq 1} \lambda^T g$$

then expressions for $\partial\|g^+\|$ can be obtained as

$$\partial\|g^+\| = \{\lambda \mid \lambda^T g = \|g^+\|,\ \lambda \geq 0,\ \|\lambda\|_D \leq 1\}$$

The dual of the l_1 norm is the l_∞ norm and the dual of the l_∞ norm is the l_1 norm. The l_2 norm is self dual.

Applying Theorem 2.5.1 to *EPF* it is possible to obtain optimality conditions for *EPF* which can be written as

$$\left.\begin{array}{l} 0 = \nu\nabla f^* + \sum_{i\in I} \lambda_i^* a_i^* \\ 0 \leq \lambda_i^* \leq 1 \\ g_i^* > 0 \ \Rightarrow\ \lambda_i^* = 0 \\ g_i^* < 0 \ \Rightarrow\ \lambda_i^* = 1 \end{array}\right\} \ i \in I$$

In order to prove equivalence of *NLP* and *EPF* it is necessary to consider a second order sufficient condition for a local minimizer of the exact penalty problem *EPF*. This second order sufficient condition is given in the following Theorem.

Theorem 2.5.2 *If there exist multipliers $\lambda^* \in \partial h(g^*)$ such that the first order necessary conditions for* EPF*,*

$$0 = \nu\nabla f^* + A^*\lambda^*$$

hold and if in addition

$$s^T\nabla^2\mathcal{L}(x^*,\lambda^*)s > 0\ ,\ \forall s \in G^*$$

then x^* is a strict local minimizer of $\Phi(x)$, where the Lagrangian is defined as

$$\mathcal{L}(x,\lambda) = \nu f(x) + \lambda^T g(x)$$

and G^* is the set of all normalized directions of zero slope at x^*, that is

$$G^* = \{s \mid \max_{\lambda\in\partial h^*} s^T(\nabla f^* + A^*\lambda) = 0,\ \|s\|_2 = 1\}$$

Now it is possible to examine the equivalence of *NLP* and *EPF*. If the penalty parameter is small enough, that is, if

$$\nu < \frac{1}{\|\lambda^*\|_D}$$

where λ^* are the optimal multipliers of *NLP*, then the solution x^* of *EPF* also solves *NLP*, provided that a second order sufficient condition holds at x^* for *NLP* and *EPF*. This implies that, at least in theory, it is possible to solve *NLP* by a single application of an unconstrained minimizer to *EPF*. Unfortunately, λ^* is *not* known a priori and it may take several "guesses", before x^* is obtained. More importantly though, Φ^* is usually a nonsmooth function and this poses additional problems. Many optimization packages use a penalty function indirectly, as a merit function, to decide the acceptance of a step.

Chapter 3

Branch–and–Bound in Integer Programming

3.1 Introduction

One of the most successful methods for solving MINLP problems is branch–and–bound. It is also the oldest general MINLP solver, dating back to Dakin [13]. All other methods presented in the subsequent chapters rely on branch–and–bound to solve the respective master program relaxations. This chapter presents a general framework algorithm for nonlinear branch–and–bound and discusses, in particular, issues arising in MIQP branch–and–bound.

The introduction of MIQP master programs in Chapter 5 has motivated a detailed study of an MIQP branch–and–bound algorithm. However, MIQP problems are not only interesting as master problems in the aforementioned algorithms but also in their own right, having various applications such as index tracking for passive portfolio management [46], the optimal sizing and siting of substations in a network routing problem and so on.

Unlike MILP problems for which a large amount of literature exists and general as well as special purpose algorithms are available, MIQP problems have not been investigated to the same extent. In order to solve the MIQP master problems it therefore becomes necessary to develop an MIQP solver. It is important to realise that an approach to MIQP problems based on Benders Decomposition as suggested by Lazimy [49] is inadequate since Benders Decomposition essentially linearizes the problem and therefore encounters the same problems as linear outer approximation when curvature information is present (c.f. Chapter 5 for a detailed discussion of these difficulties). This implies that the difficulties caused by nonlinearities in outer approximation would simply arise at the MIQP level.

Section 3.2 gives the general nonlinear branch–and–bound framework and addresses some of the questions arising from it by reviewing branching rules that have been suggested. Section 3.3 gives an introduction to the dual active set method of Goldfarb and Idnani [31] in terms of a linear complementarity tableau, introducing the terminology and notation employed in the derivation of improved lower bounds for the MIQP branch–and–bound algorithm. This general method to derive improved lower bounds is presented in Section 3.4, and it is shown how these lower bounds can be obtained efficiently. Section 3.5 presents a small example for the new improved bounds of Section 3.4 and gives some numerical results.

The aim of this chapter is not to provide an MIQP solver which works *best* under *all* possible circumstances but rather a routine that works well on general MIQP problems (including pure integer problems) such as those generated by the outer approximation algorithm of Chapter 5. There is no doubt that structure in the MIQP – such as the structure underlying the quadratic

assignment or the quadratic knapsack problem – should be exploited when writing an algorithm whenever possible. Nevertheless, the ideas presented in Section 3.4 will still be useful in many special cases.

3.2 The branch–and–bound methodology

Branch–and–bound is a general framework for solving integer and combinatorial problems. The combinatorial part of the problem (determining the optimal integer assignment) is solved by a tree search in which NLP relaxations of the MINLP problem are solved and non–integer NLP–solutions are eliminated by adding simple bounds (branching). By using lower and upper bounds it is possible to limit the tree search, thus avoiding complete enumeration. It is possible to interpret branch–and–bound as a clever rounding procedure which aims to produce an integer solution by "rounding" the fractional solution of the NLP relaxation.

The present section largely follows Beale's [5] paper on MILP branch–and–bound presenting it as a tree search. The methodology is presented for general MINLP problems and whenever a remark applies only to MIQP problems this is indicated.

The particular problem which is considered in this section is

$$\mathcal{P} \begin{cases} \min\limits_{x} & f(x) \\ \text{subject to} & g(x) \leq 0 \\ & x \in X \ , \ x_i \text{ integer } \forall \ i \in I \end{cases}$$

where f and g are continuously differentiable convex functions and it is assumed that the feasible region is bounded, which can be achieved for instance by adding simple bounds on the variables.

It is important to realise, that in general $\mathcal{P}$ cannot be solved like an NLP problem. This is due to the integer restrictions on the variables. Even verifying that a given vector x^* solves $\mathcal{P}$ is not easy, since no optimality conditions exist for $\mathcal{P}$ that can be checked. Branch–and–bound gives a framework in which $\mathcal{P}$ can be solved.

In order to describe a branch–and–bound algorithm it is necessary to introduce some notation and terminology. Let $\mathcal{P}'$ denote the problem obtained from $\mathcal{P}$ by relaxing all integer restrictions. Problem $\mathcal{P}'$ is then an ordinary NLP. The branch–and–bound algorithm starts by solving $\mathcal{P}'$, giving x' as its solution. If x' satisfies all integer restrictions it is said to be an *integer feasible solution* and in this case also solves $\mathcal{P}$ and the algorithm stops. Otherwise there exists a variable $x'_j, \ j \in I$ which does not take an integer value (and is said to be *fractional*). Branch–and–bound then proceeds by *branching* on a fractional variable, x'_j say. This is done by introducing two new subproblems obtained from $\mathcal{P}'$ by adding the simple bound

$$x_j \geq [x'_j + 1]$$

to one and

$$x_j \leq [x'_j]$$

to the other, where $[a]$ denotes the largest integer not greater than a. The solution of $\mathcal{P}$ is contained in the feasible region of one of the two new subproblems and the process can be repeated.

The Branch–and–bound algorithm continues to solve and generate new subproblems and this is best described as a tree search. The nodes of the tree correspond to NLP subproblems, represented by $\circ$. In general a node is an NLP problem in which additional simple bounds due to the branching

have been added to $\mathcal{P}'$. A node can thus be expressed as

$$\mathcal{P}(l,u)\begin{cases} \min\limits_{x} & f(x) \\ \text{subject to} & g(x) \leq 0 \\ & x \in X \\ & l \leq x \leq u \end{cases}$$

where $l \leq u$. The root of the tree corresponds to $\mathcal{P}' = \mathcal{P}(-\infty, \infty)$.

If the solution to a node is not integer feasible then two new nodes $\mathcal{P}(l^+, u^+)$ and $\mathcal{P}(l^-, u^-)$ are generated by branching on say x_j.

$$\begin{array}{lll} l_j^+ = [x_j + 1] & & u_j^- = [x_j] \\ l_i^+ = l_i & \forall i \neq j & u_i^- = u_i \quad \forall i \neq j \\ u_i^+ = u_i & \forall i & l_i^- = l_i \quad \forall i \end{array}$$

The new nodes are sometimes called *child–nodes* and the original node is called the *parent–node*. The optimal value of the parent problem provides a lower bound on the optimal value of the child problems and how this is exploited in the branch–and–bound procedure is explained in the next paragraph. A *pending node* is a node which has not yet been solved. The branch–and–bound algorithm searches the tree until no pending nodes remain.

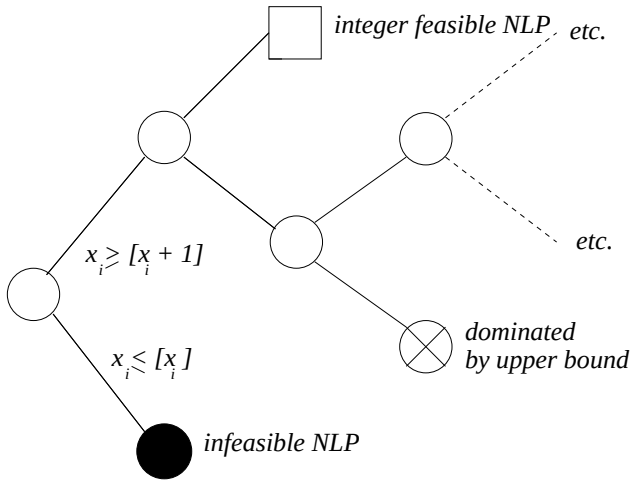

Figure 3.1: Typical Branch–and–Bound tree

It is not always necessary to search the complete tree and the success of branch–and–bound is partly due to the fact that whole subtrees can be excluded from further consideration by *fathoming* their respective root node. A node can be considered to be fathomed if it meets any of the following criteria.

- If a node is infeasible, then any node in its subtree is also infeasible and the node can be assumed to be fathomed (indicated by a •).
- If a node produces an integer feasible solution then this solution is optimal in the whole subtree starting at this node. And this node can therefore be assumed to be fathomed (indicated by □ in the tree).
  The optimal value of this node provides an *upper bound* on the solution of P, which can be used to fathom nodes.

- If the optimal value of a node or its lower bound is greater than or equal to the current upper bound then this node is fathomed since no improved solution can be found in its subtree (indicated by a $\otimes$ in the tree).

A typical branch–and–bound tree is illustrated by Figure 3.1

The tree structure is best represented by a stack on the computer together with some mechanism for reordering the stack. (Strictly speaking this is not a stack, although the term has been used widely in this context.) With every problem on the stack it is necessary to store the lower and upper bounds on the integer variables, an advanced basis, an estimate E for its solution and a guaranteed lower bound L. Initially $\mathcal{P}'$ is placed on the stack and a general step of the algorithm can be described as follows.

Algorithm: Branch–and–bound

initialization: Set $f^* = \infty$ as upper bound.

WHILE (stack is not empty) **DO BEGIN**

- Remove a problem from the stack and solve it.
  Let the solution be x' and $f' = f(x')$.
  IF (no solution exists) **THEN**
  This node can be assumed to be fathomed.

- **IF** ($f' \geq f^*$) **THEN**
  Abandon this problem, no improved solution can be obtained.
  ELSEIF (x' is integer) **THEN**
  Update the current best point by setting
  $f^* = f'$, $x^* = x'$
  Remove all problems from the stack with lower bounds L greater or equal f^*.
  ELSE
  Branch on an integer variable that takes a fractional value and add two new problems to the stack.

END (WHILE)

This description of the algorithm leaves open a number of important questions which are addressed in the remainder of this section.

Q1 How should the nodes/subproblems be solved ?

Q2 Which problem should be solved next ?

Q3 On which variable should the algorithm branch ?

Q4 How can improved lower bounds be computed ?

Questions **Q3** and **Q4** make a local decision about one particular node and these are often referred to as *tactical* decisions. Question **Q2** concerns a selection of nodes and is referred to as a *strategic* decision. The success of a branch–and–bound algorithm depends crucially on the quality of the tactical and strategic decisions made during the tree search.

Question **Q1** concerns the type of method that should be used to solve the NLP subproblems. In the special case where $\mathcal{P}$ is an MIQP problem, then using a dual method to solve the subproblems offers the most straightforward way to exploit the structure which the branching introduces

into the problem. Branching on a variable introduces an infeasibility for which a dual feasible point exists, namely the solution of the parent problem. A dual active set method could therefore take immediate advantage of a feasible starting point. More generally though, a primal method with different degrees of warm and hot starts can also take advantage of an advanced basis. The present implementation of an MIQP branch–and–bound algorithm uses a primal active set method with six different degrees of warm starts to solve the QP subproblems. This solver has the additional advantage that it implements a technique to resolve degeneracy and thus gives guaranteed termination even in the presence of round–off errors [18]. This guarantee is vital to enable the development of a robust MIQP code.

The strategic decision concerning the choice of the problem to be solved next (Question **Q2**) is resolved in favour of a depth–first–search of the tree with the additional feature that backtracking is done to the most promising node. The aim is to quickly find an integer feasible solution so that the resulting upper bound can be used to fathom nodes. This is implemented as a LIFO stack with a facility that allows the stack to be reordered once an infeasible or an integer feasible node has been reached.

Many authors suggest a branching rule (Question **Q3**) based on the importance of the integer variables (see [5] for MILP and [36] for MINLP). This rule exploits user defined priorities which are assigned to the integer variables, and branches on the most important variable first. Variables which should usually be given a high priority are binary variables, among which special ordered set variables are often very important in the application.

A branching rule which does not require any problem specific knowledge but is based on pseudo–costs has been introduced by Benichou, Gauthier, Girodet, Hentges, Ribiere and Vincent [7]. Here estimates on the change in the objective function are computed by using average incremental costs [5] of increasing/decreasing x_j from the non-integer value x'_j. Although these average incremental costs depend on the particular node they are only computed once during the tree search and assumed to be constant thereafter. If these costs are denoted by F^+ and F^- respectively than

$$D_j^+ = F^+ \cdot p_j^+$$

$$D_j^- = F^- \cdot p_j^-$$

are the average costs involved with introducing the constraints

$$x_j \geq [x'_j + 1]$$

and

$$x_j \leq [x'_j]$$

where $p_j^+ = 1 - \phi_j$, $p_j^- = \phi_j$ and $\phi_j = x'_j - [x'_j]$ is the fractional part of x'_j. The strategy selects the integer variable which maximizes the smaller of its associated pseudo–costs, that is

$$\max_j \{ \min(D_j^+ \ , \ D_j^-) \}$$

Gupta and Ravindran report experience with pseudo costs for a range of MINLP test problems and conclude that it is inferior to the branching rule which is described in the next paragraph.

A very simple branching rule which has proved successful both for MINLP branch and bound [36] and the Travelling Salesman Problem [58] is based on the amount by which the variables violate the integer restrictions. This strategy selects the integer variable which is furthest from its nearest integer value, i.e.

$$\max_j \{ \min(p_j^+ \ , \ p_j^-) \}$$

This branching rule is referred to as *maximal fractional part branching* by Breu and Burdet [9]. The aim is to obtain the largest increase in the objective function so that more nodes can be pruned later on.

The remainder of this section studies the case where $\mathcal{P}$ is an MIQP. In this case f and g possess special structure and

$$f(x) = \tfrac{1}{2}x^T Gx + g^T x$$

$$g(x) = A^T x - b$$

where $G \in \mathbb{R}^{n\times n}$ is a symmetric and positive semi–definite matrix, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times m}$ and $b \in \mathbb{R}^m$. A survey of methods for MIQP problems is given by Volkovich et. al. [75]. Here only the branching rule suggested by Körner is considered.

Körner [45] proposes a branching rule for MIQP problems based upon minimizing the size of the tree so that fewer QP problems need be solved. In [45] a branch on x_j means the introduction of a series of problems in which x_j is fixed at

$$\ldots, [x_j - 1], [x_j], [x_j + 1], \ldots$$

Körner then defines the level set L^* of feasible points x whose objective value is less than or equal f^*, the unknown optimal value of the MIQP.

$$L^* = \{x : f(x) \leq f^* : A^T x \leq b,\ x \geq 0\}$$

He then defines projections of L^* onto the i^{th} coordinate axis, denoted by L^i,

$$L^i = \{x : \exists\, y \in L^* \text{ such that } x = e_i e_i^T y\}.$$

The branching rule is then to branch on the integer variable x_j that satisfies

$$card(L^j \cap \mathbf{Z}) \leq card(L^i \cap \mathbf{Z})\ \forall i$$

where $\mathbf{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ and $card(S)$ is the cardinality of the set S. Körner argues that such a choice minimizes the size of the tree. He shows how estimates for $card(L^i \cup \mathbf{Z})$ can be derived efficiently from the final Linear Complementarity tableau (see e.g. [18]) of the parent QP.

Körner's rule can be seen as implementing the rule based upon the importance of the variables. It has been suggested that binary variables should be given a high priority. This seems to be supported by this branching rule, since $card(L^i \cap \mathbf{Z}) \leq 2$ for binary variables.

3.3 Linear Complementarity Problems and the dual active set method

This section describes the dual active set method of Goldfarb and Idnani [31] in terms of a linear complementarity tableau, thereby introducing the notation and terminology employed in the next section to derive improved lower bounds for the MIQP branch–and–bound algorithm. First the linear complementarity problem (LCP) is introduced and related to the more efficient null space formulation of a QP problem. Next the LCP tableau is used to explain the dual active set method (ASM). The presentation in this section follows largely Fletcher's exposition [18].

The problem of interest here is

$$(QP)\begin{cases} \min\limits_{x} & f(x) = \frac{1}{2}x^T Gx + g^T x \\ \text{subject to} & A^T x - b \geq 0 \\ & x \geq 0,\ x \in \mathbb{R}^n \end{cases}$$

Any problem that is generated as a node during the branch–and–bound tree search can be written in the above format. Introducing multipliers λ for the general constraints, multipliers π for the simple bounds and residuals or slack variables $r = A^T x - b$, the Kuhn–Tucker conditions can be written as

$$Gx + g - A\lambda - \pi = 0 \tag{3.1}$$
$$A^T x - b - r = 0 \tag{3.2}$$
$$x, \pi, \lambda, r \geq 0 \tag{3.3}$$
$$\pi^T x = 0, \lambda^T (A^T x - b) = \lambda^T r = 0. \tag{3.4}$$

The equalities (3.1) and (3.2) can be written in matrix form

$$\begin{bmatrix} I & 0 & -G & A \\ 0 & I & -A^T & 0 \end{bmatrix} \begin{pmatrix} \pi \\ r \\ x \\ \lambda \end{pmatrix} = \begin{pmatrix} g \\ -b \end{pmatrix} \tag{3.5}$$

and this can be written in a more compact *tableau* format, where the leading identity matrix is omitted.

$$\begin{array}{c} \\ \pi \\ r \end{array} \begin{array}{c} \begin{array}{cc} x & \lambda \end{array} \\ \begin{bmatrix} -G & A \\ -A^T & 0 \end{bmatrix} \end{array} \begin{bmatrix} g \\ -b \end{bmatrix}$$

Equation (3.5) can be interpreted as defining *basic* or *dependent* variables (π, r) in terms of *nonbasic* or *independent* variables (x, λ). The value of the nonbasic variables is zero and the value of the basic variables is given by the right hand side of the tableau $(g, -b)$. A tableau is called *complementary*, if basic and nonbasic variables satisfy the *complementarity condition* (3.4). A tableau that is complementary is referred to as a *Linear Complementarity Tableau*. By applying row operations it is possible to change the partitioning of the variables into basic and nonbasic (by making the corresponding column a unit column). The aim is to obtain a partition that is complementary with a non–negative right hand side, since such a partitioning corresponds to a Kuhn–Tucker point of the QP.

For example, in order to obtain a dual feasible partition $(\pi \geq 0, \lambda \geq 0)$, x and r can be chosen as nonbasic variables, provided the Hessian G is positive definite. The new tableau is obtained by taking the system (3.5) and block–pivoting on G. The new tableau is thus given by

$$\begin{array}{c} \\ x \\ r \end{array} \begin{array}{c} \begin{array}{cc} \pi & \lambda \end{array} \\ \begin{bmatrix} -G^{-1} & -G^{-1}A \\ A^T G^{-1} & -A^T G^{-1} A \end{bmatrix} \end{array} \begin{bmatrix} -G^{-1} g \\ -b - A^T G^{-1} g \end{bmatrix}$$

This tableau corresponds to $x = -G^{-1}g$, the unconstrained minimizer of f. However, this is not the only way to obtain a dual feasible tableau and often a common phase I approach might be more appropriate.

In order to describe the pivoting process in more detail it is convenient to simplify the notation and not distinguish between x and r or π and λ, but merely between active and inactive variables. This can be achieved by including the simple bounds in the general constraints and considering the following QP

$$(QP) \begin{cases} \min\limits_{x} & f(x) = \frac{1}{2} x^T G x + g^T x \\ \text{subject to} & A^T x - b \geq 0 \end{cases}$$

Now r refers to all constraint residuals including the previous x and λ similarly refers to all multipliers including the multipliers of the simple bounds π. In an active set method the constraints are divided into two sets. The *active set* $\mathcal{A}$ and the set $\mathcal{I}$ indexing the remaining constraints. If no degeneracy occurs, then the active set $\mathcal{A}$ can be identified with the set of active constraints. It turns out that the basic/nonbasic variables in the tableau are closely related to the sets $\mathcal{A}$ and $\mathcal{I}$. Partitioning λ and r into active and inactive parts indexed by $\mathcal{A}$ and $\mathcal{I}$ respectively, the nonbasic variables can be identified with $\lambda_{\mathcal{I}}$ and $r_{\mathcal{A}}$, the multipliers of the inactive constraints which are zero by complementarity and the residuals of the active constraints respectively. The basic variables on the other hand can be identified with $r_{\mathcal{I}}$ and $\lambda_{\mathcal{A}}$, the residuals of the inactive constraints and the multipliers of active constraints.

In general, the corresponding LCP tableau then has the form

$$\begin{array}{cc} & \begin{array}{cc} r_{\mathcal{A}} & \lambda_{\mathcal{I}} \end{array} \\ \begin{array}{c} \lambda_{\mathcal{A}} \\ r_{\mathcal{I}} \end{array} & \left[\begin{array}{cc} G' & A' \\ -A'^T & H' \end{array}\right] \end{array} \quad \left[\begin{array}{c} \lambda'_{\mathcal{A}} \\ r'_{\mathcal{I}} \end{array}\right]$$

where the current value of the basic variables is given by the right hand side $(\lambda'_{\mathcal{A}}, r'_{\mathcal{I}})$.

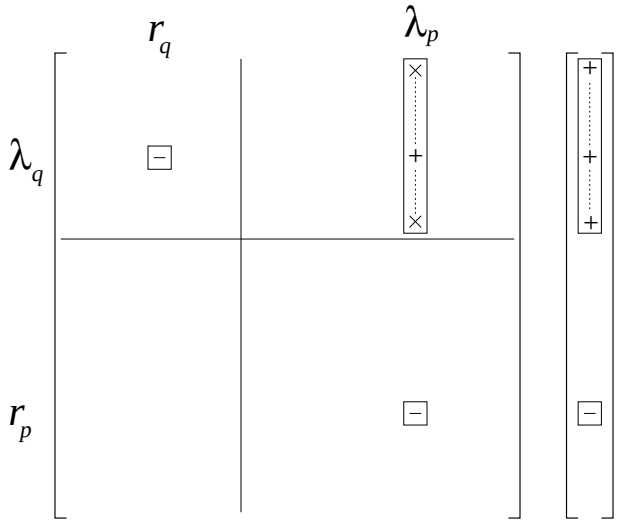

Figure 3.2: LCP tableau for dual ASM with relevant entries

In the dual ASM, the multipliers of the active constraints are non–negative (i.e. $\lambda_{\mathcal{A}} \geq 0$) and the tableau is optimal as soon as primal feasibility is obtained (i.e. $r_{\mathcal{I}} \geq 0$). In general there exists an $r'_p < 0$ for some $p \in \mathcal{I}$ and the dual ASM aims to drive all such negative residuals to zero whilst maintaining dual feasibility and complementarity. If $r'_p < 0$, then a ratio test determines which λ_q for $q \in \mathcal{A}$ is driven to zero by an increase α in r'_p. If no ratios exists, then the dual is unbounded and the algorithm stops with the message that the primal is infeasible.

Otherwise the tableau is updated by subtracting α times the column corresponding to λ_p from the right hand side. Two different pivot operations are then possible. Either λ_q is driven to zero, that is λ_q becomes nonbasic and r_q basic, or r_p is driven to zero before λ_q reaches zero, that is λ_p becomes basic. In the first case, (λ_q, r_q) is the pivot element which adds a column to G', and in the second case (λ_p, r_p) is the pivot element which removes one column from G'. The two pivot operations can also be interpreted as updates to the active set in the following way. In the first case, $\lambda_q \searrow 0$, q is dropped from the active set, r_p is not driven to feasibility and the process is

repeated (minor iteration). In the second case, $r_p \nearrow 0$ a new constraint p is added to the active set, removing a primal infeasibility and this ends a major iteration The relevant entries are highlighted in the tableau in Figure 3.2.

A detailed description of the dual active set method in tableau form is given below.

Dual Active Set Method (Tableau formulation)

LOOP

1. *Optimality Test:*
   IF ($r_{\mathcal{I}} \geq 0$) **THEN STOP** (tableau is optimal)
   ELSE find $r'_p < 0$ and get λ_p column of the tableau.

2. *Ratio Test:*
   Calculate pertinent ratios of boxed elements (only for elements in λ_p column that are positive and for H element if it is negative, as given in Figure 3.2). That is

$$\alpha = \min_{i:-A'_{pi}>0} \frac{-A'_{pi}}{\lambda'_i},$$

   where $-A'_p$ is the column of $-A'$ corresponding to λ_p.
   IF (no ratios) **THEN STOP** (dual is unbounded).

3. *Update:*
   Increase λ_p by α and subtract $\alpha \cdot (\lambda_p$–column) from the right hand side column. That is

$$\lambda_p = \alpha \qquad \begin{array}{l} \lambda'_{\mathcal{A}} = \lambda'_{\mathcal{A}} - \alpha A'_p \\ r'_{\mathcal{I}} = r'_{\mathcal{I}} - \alpha H'_p \end{array}$$

4. *Pivot:*
   (a) **IF** ($\lambda_q \searrow 0$) **THEN** (minor iteration)
   pivot on (λ_q, r_q), removing q from $\mathcal{A}$, get the new column λ_p and **GOTO 2**.
   (b) **IF** ($r_p \nearrow 0$) **THEN** (end major iteration)
   pivot on (λ_p, r_p), adding constraint p to $\mathcal{A}$.

ENDLOOP

The major iterations are repeated until an optimal tableau is encountered or infeasibility is detected. The two different kinds of pivot are illustrated in Figure 3.3. Type (a) pivots on a diagonal element of G' and corresponds to removing an active constraint from the active set. The dotted line indicates the partitioning after the pivot step. Type (b) pivots on a diagonal element of H' which corresponds to adding a new constraint to $\mathcal{A}$.

In order to obtain expressions for G', H' and A', the equality problem corresponding to a given active set $\mathcal{A}$ is considered. This problem is derived from QP by (temporarily) fixing the constraints in the active set as equalities, that is

$$(EP) \begin{cases} \min\limits_{x} & f(x) = \frac{1}{2}x^T G x + g^T x \\ \text{subject to} & r_{\mathcal{A}} = A_{\mathcal{A}}^T x - b_{\mathcal{A}} = 0 \end{cases}$$

Problem EP is much easier to solve than QP and the active set method aims at modifying the active set until a Kuhn–Tucker point for EP is found which is also a Kuhn–Tucker point for QP. A solution to EP can be found by solving the following linear system obtained by the Method of

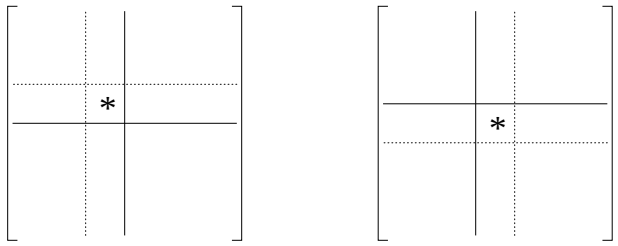

Figure 3.3: Different types of pivot

Lagrange Multipliers (e.g. [17]).

$$\begin{bmatrix} G & -A_{\mathcal{A}} \\ -A_{\mathcal{A}}^T & 0 \end{bmatrix} \begin{pmatrix} x' \\ \lambda'_{\mathcal{A}} \end{pmatrix} = - \begin{pmatrix} g \\ b_{\mathcal{A}} \end{pmatrix}$$

Expressing the inverse of the matrix above as

$$\begin{bmatrix} G & -A_{\mathcal{A}} \\ -A_{\mathcal{A}}^T & 0 \end{bmatrix}^{-1} = \begin{bmatrix} H & -T \\ -T^T & U \end{bmatrix} \tag{3.6}$$

it becomes straightforward to solve EP. In practice, however, the inverse is not obtained explicitly but is used implicitly. There are two alternative methods for the representation of the inverse: the range space formulation (e.g. [18]) and the null space formulation. Only the latter of the two is relevant to the present work since it is implemented in the underlying QP solver. In the null space formulation the matrix $A_{\mathcal{A}}$ is bordered by an arbitrary matrix V such that the inverse

$$[A_{\mathcal{A}} : V]^{-1} = \begin{bmatrix} Y^T \\ Z^T \end{bmatrix}$$

exists. The virtues of the different choices are explained in Fletcher [17]. In practice the inverse is not computed explicitly and the QP–code that is being used in the branch–and–bound algorithm uses LU–factors of the matrix rather than the inverse. With the definition of Y and Z it is possible to give explicit formulas for H, T and U.

$$\begin{aligned} H &= ZM^{-1}Z^T \\ T &= Y - ZM^{-1}Z^TGY \\ U &= Y^TGZM^{-1}Z^TGY - Y^TGY \end{aligned}$$

where $M = Z^TGZ$ is the reduced Hessian matrix. Again, instead of calculating the inverse of M explicitly the QP–solver computes LDL^T–factors of it.

Now it is possible to give expressions for G', H' and A' in terms of H, T, U, G and A and thus in terms of the original data of the problem and its corresponding factorizations. The Kuhn–Tucker conditions for QP and the definition of r give

$$\begin{aligned} Gx + g &= A\lambda \\ r &= A^Tx - b \end{aligned}$$

In order to eliminate x from these equations, it is convenient to express them in terms of perturbations from their current values (λ' and r'), that is

$$\begin{aligned} G(x - x') &= A(\lambda - \lambda') \\ r - r' &= A^T(x - x') \end{aligned}$$

Splitting $A = [A_{\mathcal{A}} : A_{\mathcal{I}}]$ and r and λ accordingly the following equations are obtained

$$G(x - x') = A_{\mathcal{A}}(\lambda_{\mathcal{A}} - \lambda'_{\mathcal{A}}) + A_{\mathcal{I}}(\lambda_{\mathcal{I}} - \lambda'_{\mathcal{I}}) \tag{3.7}$$

$$r_{\mathcal{A}} - r'_{\mathcal{A}} = A_{\mathcal{A}}^T(x - x') \tag{3.8}$$

$$r_{\mathcal{I}} - r'_{\mathcal{I}} = A_{\mathcal{I}}^T(x - x') \tag{3.9}$$

complementarity implies that $r'_{\mathcal{A}} = 0$ and $\lambda'_{\mathcal{I}} = 0$, so that equations (3.7) and (3.8) can be arranged as

$$\begin{bmatrix} G & -A_{\mathcal{A}} \\ -A_{\mathcal{A}}^T & 0 \end{bmatrix} \begin{pmatrix} x - x' \\ \lambda_{\mathcal{A}} - \lambda'_{\mathcal{A}} \end{pmatrix} = \begin{pmatrix} A_{\mathcal{I}}\lambda_{\mathcal{I}} \\ -r_{\mathcal{A}} \end{pmatrix}$$

This system is now inverted using (3.6) to obtain an expression for $x - x'$ in λ and r only.

$$\begin{aligned} x - x' &= HA_{\mathcal{I}}\lambda_{\mathcal{I}} + Tr_{\mathcal{A}} \\ \lambda_{\mathcal{A}} - \lambda'_{\mathcal{A}} &= -T^T A_{\mathcal{I}}\lambda_{\mathcal{I}} - Ur_{\mathcal{A}} \end{aligned}$$

The last equation defines the basic variables $\lambda_{\mathcal{A}}$ in terms of the nonbasic variables $\lambda_{\mathcal{I}}$ and $r_{\mathcal{A}}$. In order to obtain the equation that defines $r_{\mathcal{I}}$ in terms of $\lambda_{\mathcal{I}}$ and $r_{\mathcal{A}}$ in the tableau, $x - x'$ is substituted in (3.9) to give

$$r_{\mathcal{I}} - r'_{\mathcal{I}} = A_{\mathcal{I}}^T HA_{\mathcal{I}}\lambda_{\mathcal{I}} + A_{\mathcal{I}}^T Tr_{\mathcal{A}}$$

and this can be rearranged to give the tableau

$$\begin{array}{c} \\ \lambda_{\mathcal{A}} \\ r_{\mathcal{I}} \end{array} \begin{array}{c} \begin{array}{cc} r_{\mathcal{A}} & \lambda_{\mathcal{I}} \end{array} \\ \begin{bmatrix} U & T^T A_{\mathcal{I}} \\ -A_{\mathcal{I}}^T T & A_{\mathcal{I}}^T HA_{\mathcal{I}} \end{bmatrix} \end{array} \begin{bmatrix} \lambda'_{\mathcal{A}} \\ r'_{\mathcal{I}} \end{bmatrix}$$

And expressions for G', A' and H' can be readily obtained, which are summarised in the following theorem [18].

Theorem 3.3.1 *Let $\mathcal{A}$ be the active set that arises in an ASM, such that the matrix $A_{\mathcal{A}}$ has full rank and the resulting EP has a unique solution x'. The corresponding complementarity tableau then has the form*

$$\begin{array}{c} \\ \lambda_{\mathcal{A}} \\ r_{\mathcal{I}} \end{array} \begin{array}{c} \begin{array}{cc} r_{\mathcal{A}} & \lambda \end{array} \\ \begin{bmatrix} G' & A' \\ -A'^T & H' \end{bmatrix} \end{array} \begin{bmatrix} \lambda'_{\mathcal{A}} \\ r'_{\mathcal{I}} \end{bmatrix} \tag{3.10}$$

where

1. *$A' = T^T A_{\mathcal{I}}$ and the off–diagonal blocks in (3.10) are skew–symmetric,*
2. *$G' = U$ is symmetric and negative definite if G is positive definite,*
3. *$H' = -A_{\mathcal{I}}^T HA_{\mathcal{I}}$ is symmetric and negative semi definite.*

Points 1. to 3. allow methods based on the complementarity tableau to be implemented efficiently as null space methods. The case where $|\mathcal{A}| = n$ is also relevant for the next section and expressions for G', H' and A' are given by

$$\begin{aligned} G' &= -A_{\mathcal{A}}^{-1} G A_{\mathcal{A}}^{-T} \\ A' &= A_{\mathcal{A}}^{-1} A_{\mathcal{I}} \\ H' &= 0. \end{aligned}$$

The next section shows, how lower bounds for child nodes can be derived in a branch–and–bound process by taking a single step of the dual ASM. The procedure is described in terms of the complementarity tableau introduced in this section. Since branching introduces only a single primal infeasibility an initial dual feasible tableau for the child nodes is readily available through the optimal tableau of the parent node and this is exploited in deriving the lower bounds.

3.4 Improved lower bounds for MIQP branch–and–bound

Lower bounds for the problems that are placed on the stack in branch and bound are readily available as the optimal value of the parent problem, if $\mathcal{P}$ is convex. However, these bounds are usually very weak and rarely give rise to any fathoming at a later stage once an integer feasible solution has been obtained. Improved, guaranteed lower bounds have therefore been considered in [5] for MILP and are readily derived in the context of an Active Set Method (ASM) (e.g. [17]) for MILP problems.

This section considers MIQP problems and shows how improved lower bounds can be derived for the child nodes of $\mathcal{P}'$, where $\mathcal{P}'$ is the QP relaxation of $\mathcal{P}$ as defined in Section 3.2

$$\mathcal{P}' \left\{ \begin{array}{ll} \min\limits_{x} & f(x) = \frac{1}{2} x^T G x + c^T x \\ \text{subject to} & A^T x \geq b \\ & x \geq 0. \end{array} \right.$$

The procedure can be readily applied to any other node of the MIQP branch–and–bound tree. Let the solution to $\mathcal{P}'$ be x' and assume that $x'_j \ , \ j \in I$ is fractional. Define $\mathcal{P}^-$ as the QP obtained from $\mathcal{P}'$ by branching down (i.e. by adding the simple bound $x_j \leq [x'_j]$) and $\mathcal{P}^+$ as the QP obtained from $\mathcal{P}'$ by branching up (i.e. by adding $x_j \geq [x'_j + 1]$).

The way in which lower bounds can be derived depends on the number of active constraints at the solution of $\mathcal{P}'$. If fewer than n constraints are active then curvature contributes to the minimizer x'. In this case it is shown in Section 3.4.1 that lower bounds can be obtained by taking one step of the dual ASM. If, on the other hand, n constraints are active at the solution of $\mathcal{P}'$ then the bounds are derived in the same way as lower bounds for MILP branch–and–bound and this is discussed in Section 3.4.2. An implementation of the methods described for a primal ASM solver for QP problems is presented in Section 3.4.3.

3.4.1 Improved lower bounds for less than n active constraints

First it is described how the dual ASM can be used to derive improved lower bounds for the case that less than n constraints are active at the solution of the parent problem $\mathcal{P}'$. The dual ASM of Goldfarb and Idnani starts with a dual feasible point, i.e. a vector satisfying the Kuhn–Tucker conditions except for primal feasibility. This makes the method very amenable to MIQP branch–and–bound since upon solving the parent problem its solution (x', λ') is dual feasible in both child problems. (The branching simply adds one primal infeasibility to the parent problem.) The dual ASM takes one constraint that is violated and tries to increase its multiplier thus reducing the

primal infeasibility whilst maintaining all other feasibilities. In order to derive an improved lower bound one step of the dual ASM is performed.

The only constraint that is violated in $\mathcal{P}^-$ and $\mathcal{P}^+$ is the newly added simple bound on x_j. The method that is described here can be seen as moving parametrically from x' towards satisfying the new constraint whilst maintaining dual feasibility. The resulting point is dual feasible and a lower bound is therefore obtained by evaluating the Lagrangian of $\mathcal{P}^-$ (or $\mathcal{P}^+$) at this point.

The step of the dual ASM is not done explicitly but all necessary quantities can be computed from the final LCP tableau of the parent problem. This tableau is either available directly or indirectly (as in our case) if the parent problem is solved with an ASM. It is shown in the previous section, that the final LCP tableau of the parent problem can be written as

$$\begin{array}{cc} & \begin{array}{cc} r_{\mathcal{A}} & \lambda_{\mathcal{I}} \end{array} \\ \begin{array}{c} \lambda_{\mathcal{A}} \\ r_{\mathcal{I}} \end{array} & \left[\begin{array}{cc} G' & A' \\ -A'^T & H' \end{array}\right] \end{array} \left[\begin{array}{c} \lambda'_{\mathcal{A}} \\ r'_{\mathcal{I}} \end{array}\right]$$

Since x'_j is fractional, it follows that $j \in \mathcal{I}$, or in other words x'_j is not at a simple bound. Let $\pi_j = \lambda_j$ be the complementary variable to x_j. The dual ASM aims to increase π_j away from zero until one of the other multipliers becomes zero. This corresponds to moving x_j parametrically towards its new bound, $[x'_j]$ say. The aim is to determine how far x_j can be decreased moving towards $[x'_j]$ in the null–space of the active constraints without destroying dual feasibility.

The effect of increasing π_j on the multiplier of the active constraints is given by the column of the LCP tableau corresponding to π_j. In particular the effect on the multipliers of the active constraints is

$$\lambda_{\mathcal{A}} = \lambda'_{\mathcal{A}} - \alpha a'_j$$

for moving x_j towards $[x'_j + 1]$ and

$$\lambda_{\mathcal{A}} = \lambda'_{\mathcal{A}} + \alpha a'_j$$

for moving x_j towards $[x'_j]$, where a'_j is the column of A' corresponding to π_j. A dual ratio test determines the maximum value of α for each case.

$$\alpha^+ = \min_{a'_{ij}>0} (\frac{\lambda'_i}{a'_{ij}})$$

gives the dual feasible steplength for moving towards $[x'_j + 1]$ and

$$\alpha^- = \min_{a'_{ij}<0} (-\frac{\lambda'_i}{a'_{ij}})$$

gives the dual feasible steplength in the second case. If there exists no $a'_{ij} > 0$ or no $a'_{ij} < 0$ respectively then α is set to ∞, since the dual is unbounded. In this case no lower bound for the respective child problem needs to be computed, since it is known to be infeasible. This particular child problem is therefore not added to the stack.

In the dual active set method of Goldfarb and Idnani the step is also limited by any (inactive) constraint residual $r_{\mathcal{I}}$ being driven to zero. This is not necessary here since in order to derive an improved lower bound only a point satisfying the Kuhn–Tucker conditions except for primal feasibility is needed.

Finally the lower bounds on $\mathcal{P}^+$ and $\mathcal{P}^-$ can be computed as L^+ and L^- respectively, where

$$\begin{aligned} L^+ &= f' + \tfrac{1}{2}(\alpha^+)^2 h_j'^T G h'_j + \alpha^+(1 - \phi_j - \alpha^+ h'_{jj}) \\ L^- &= f' + \tfrac{1}{2}(\alpha^-)^2 h_j'^T G h'_j + \alpha^-(\phi_j - \alpha^- h'_{jj}). \end{aligned}$$

The lower bounds L^+ and L^- are the value of the Lagrangean at the new point. The quantity f' is the optimal value of the parent problem and the quadratic term corresponds to the second order change in the objective. In this case h'_j is the column of H' corresponding to j which is the search direction along which x_j is moved towards its new bound. The last term improves this lower bound further. It corresponds to a Lagrangean type penalty for the violation of the new simple bound, namely the product of the multiplier α of the new simple bound with the amount by which the simple bound is not satisfied.

3.4.2 Improved lower bounds for n active constraints

If on the other hand n constraints are active at the solution of $\mathcal{P}'$ then the above procedure does not yield an improved lower bound, because $H' = 0$ in this case. It is, however, possible to improve the lower bound by applying techniques used for MILP branch–and–bound to derive improved lower bounds. These techniques consider the first order change resulting from a move away from x' towards satisfying the new simple bound.

The final LCP tableau can be simplified in this case to

$$\begin{array}{c} \\ \lambda_{\mathcal{A}} \\ r_{\mathcal{I}} \end{array} \begin{array}{c} \begin{array}{cc} r_{\mathcal{A}} & \lambda \end{array} \\ \left[\begin{array}{cc} G' & A' \\ -A'^T & 0 \end{array}\right] \end{array} \left[\begin{array}{c} \lambda'_{\mathcal{A}} \\ r'_{\mathcal{I}} \end{array}\right]$$

where A' is a (nonsingular) $n \times n$ matrix. Let $A'^{-T} = [s_1, \ldots, s_n]$ so that the directions $s_1, \ldots, s_n$ are the feasible edges pointing away from x' and the multipliers $\lambda_1, \ldots, \lambda_n$ give the slope of f along these edges. Let

$$p_j^+ = \min_{q:(s_q)_j > 0} \frac{\lambda_q}{(s_q)_j}$$

and

$$p_j^- = \min_{q:(s_q)_j < 0} \frac{\lambda_q}{-(s_q)_j}$$

If f^+ denotes the optimal solution of $\mathcal{P}^+$ then it follows from the convexity of f that

$$f^+ \geq f' + \sum_{i=1}^{n} \alpha_i s_i^T \nabla f', \tag{3.11}$$

where

$$x^+ = x' + \sum_{i=1}^{n} \alpha_i s_i$$

is the unknown minimizer of $\mathcal{P}^+$. The slope of f at x' along s_i is given by λ_i so that (3.11) simplifies to

$$f^+ \geq f' + \sum_{i=1}^{n} \alpha_i \lambda_i.$$

Since the feasible edges s_i define a convex cone and x^+ lies in this cone, the coefficients α_i must be positive, so that the following inequality follows from the definition of p_j^+

$$\begin{aligned} f^+ &\geq f' + \sum_{i=1}^{n} \alpha_i (s_i)_j \frac{\lambda_i}{(s_i)_j} \\ &\geq f' + \sum_{i=1}^{n} \alpha_i (s_i)_j p_j^+. \end{aligned}$$

In particular, since x^+ solves $\mathcal{P}^+$ it is necessary that

$$\sum_{i=1}^{n} \alpha_i (s_i)_j \geq 1 - \phi_j$$

so that the lower bound

$$L^+ = f' + (1 - \phi_j) p_j^+$$

on $\mathcal{P}^+$ is finally obtained. Similarly it is possible to deduce a lower bound for $\mathcal{P}^-$ as

$$L^- = f' + \phi_j p_j^- .$$

3.4.3 Implementation of improved lower bound procedure

This section indicates how the two lower bound procedures of the previous two sections can be implemented and gives an operation count. The QP solver employed in the present implementation uses a primal active set method, where equality QP problems are solved by a null space method [18].

At the end of each solve LU factors of the matrix $[A_{\mathcal{A}} : V]$ are available, where $A_{\mathcal{A}}$ is the matrix whose columns are the active constraints and V is any matrix such that $[A_{\mathcal{A}} : V]$ is nonsingular. If

$$[A_{\mathcal{A}} : V]^{-1} = \begin{bmatrix} Y^T \\ Z^T \end{bmatrix}$$

then expressions for A' and H' can be given, where

$$H' = -A_{\mathcal{I}} Z (Z^T G Z)^{-1} Z^T A_{\mathcal{I}}$$
$$A' = (Y - Z(Z^T G Z)^{-1} Z^T G Y) A_{\mathcal{I}}.$$

Also available are LDL^T factors of the "reduced Hessian" $(Z^T G Z)^{-1}$. In the case where fewer than n constraints are active at the solution of $\mathcal{P}'$ the following computations are carried out to derive the lower bound.

1. Find column j of $[A_{\mathcal{A}} : V]^{-1}$ and compute $v = Z^T e_j$.
   (n^2 flops)

2. Use LDL^T factors of the reduced Hessian to compute $w = (Z^T G Z)^{-1} v$.
   (k^2 flops)

3. Form $v = Zw$ using the LU factors of $[A_{\mathcal{A}} : V]^{-1}$.
   (n^2 flops)

4. Compute m scalar products to form $w = -A_{\mathcal{I}}^T v$.
   (mn flops)

5. Form the matrix vector product $v = Gw$.
   (n^2 flops)

6. Find $w = Y^T v$, using the LU factors of $[A_{\mathcal{A}} : V]^{-1}$.
   (n^2 flops)

7. Carry out the ratio test to find α^+ and α^-.
   ($n + 2$ flops)

8. Compute the lower bounds L^+ and L^-.
   ($n^2 + n + 8$ flops)

Summing the individual operation counts shows that the lower bounds can be computed in $5n^2+mn+k^2+\mathcal{O}(n)$ flops. An operation count for the lower bounds in the case where n constraints are active at the solution of $\mathcal{P}'$ reveals that this procedure requires $n^2 + \mathcal{O}(n)$ flops. It is notable that both operation counts are of the same order of magnitude. Both procedures are considerably cheaper than a QP solve, but an order of magnitude more expensive than the simpler branching rules based upon the most fractional variable. Only extensive numerical testing can reveal whether or not this additional effort results in an overall reduction in CPU time for MIQP problems and this together with a small illustrative example is considered in the next section.

3.5 A numerical example and test results

It is instructive to consider applying the above procedure to a simple example:

$$\mathcal{P}\left\{\begin{array}{ll}\min\limits_{x} & x_1^2 - x_1x_2 + x_2^2 - 3x_1 \\ \text{subject to} & -x_1 - x_2 \geq -2 \\ & x \geq 0\ ,\ x_1 \text{ integer}\end{array}\right.$$

The solution of $\mathcal{P}$ with its integer restrictions relaxed is $x' = (\frac{3}{2}, \frac{1}{2})$, $f' = -\frac{99}{36}$ and the final LCP tableau is given by

$$\begin{array}{c} \\ \lambda_1 \\ x_1 \\ x_2 \end{array}\begin{array}{c} \begin{array}{ccc} r_1 & \pi_1 & \pi_2 \end{array} \\ \left[\begin{array}{c|cc} -\frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \\ \hline \frac{1}{2} & -\frac{1}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} & -\frac{1}{6} \end{array}\right] \end{array}\left[\begin{array}{c} \frac{1}{2} \\ \frac{3}{2} \\ \frac{1}{2} \end{array}\right]$$

where π_i denotes the complementary variable to x_i. Let $\mathcal{P}^-$ denote the problem obtained from $\mathcal{P}$ by adding the new upper bound $x_1 \leq 1$. To compute a lower bound on the solution of $\mathcal{P}^-$ the dual ratio test is carried out, giving

$$\alpha^- = \min(-\frac{\frac{1}{2}}{-\frac{1}{2}}) = 1.$$

The search direction in terms of the x variables is $h' = (-\frac{1}{6}, \frac{1}{6})^T$ and the step of the dual active set method moves to

$$\hat{x} = x' + \alpha^- h' = \begin{pmatrix} \frac{4}{3} \\ \frac{2}{3} \end{pmatrix}$$

which results in the improved lower bound of $L^- = -\frac{84}{36}$. This improvement corresponds to closing 80% of the gap between the solution value of $\mathcal{P}$ and $\mathcal{P}^-$ (which is $-\frac{81}{36}$). Similarly the lower bound for $\mathcal{P}^+$ can be computed as $L^+ = -2$. Thus a branch–and–bound procedure using the lower bounding property would branch to $\mathcal{P}^-$ as the more promising node. After solving this child problem, the algorithm would not need to solve $\mathcal{P}^+$ since its lower bound is greater than the current optimum. The outcome of the lower bounding procedure is thus a decrease in the number of problems that need to be solved in a branch–and–bound process. The move of the dual ASM is illustrated in Figure 3.4.

It is indicated in Section 3.4.3 that improved lower bounds are not cheaply computed and it is therefore important to carry out numerical testing to assess to what extend the savings in the number of problems solved are mirrored by savings in the CPU time. To this end three different branching rules have been implemented which make use of the improved lower bounds to different degrees. BR1 uses the branching rule based on the largest fractional integer variable and uses no

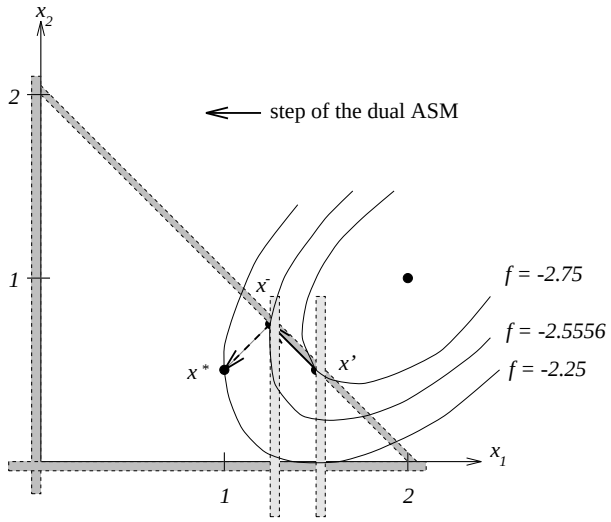

Figure 3.4: Illustration of the dual active set method

improved lower bounds at all. BR2 computes lower bounds L_j^+ and L_j^- for all fractional integer variables and branches on the integer variable that satisfies

$$\max_j \min(L_j^+, L_j^-).$$

Finally, BR3 uses branching rule BR1 to determine the branching variable, but in addition computes lower bounds for the problems that are placed on the stack.

The test problems come from a variety of backgrounds. "Beale" and "HS76" are problems found in the literature (e.g. [38]). "AVGAS" and "AFIRO" are originally LP problems from the SOL test set. Both problems where modified by adding integer restrictions and a positive definite Hessian matrix. In the case of "AFIRO" the linear objective has also been changed to give better scaled results. The second group of "AFIRO" problems has more integer variables than the first, which increased the gap between the optimum of their QP relaxation and the integer optimum. "QAP" is a small quadratic assignment problem for the optimal assignment of 4 plants to 6 locations. Finally "RANDOM 1" to "RANDOM 3" are randomly generated MIQP test problems. Table 3.1 gives a brief description of the test problems. The first column gives the name of the problem followed by four columns that detail the dimensions of the problems; n_i is the number of integer variables, n_c is the number of continuous variables, m_e gives the number of equality constraints and finally m_i details the number of inequality constraints. The last column gives comments such as the range of the integer variables x_I.

The next two tables give the results of the test run on a SUN-SPARC SLC using the FORTRAN 77 compiler with the –fast option. Table 3.2 gives the number of QP problems generated by each of the branching rules and the number of QP problems solved. Finally, Table 3.3 gives the CPU time needed by each branching rule to solve the MIQP problem. The results for the randomly generated MIQP are averages over 5 runs.

Table 3.2 indicates that with BR1 virtually no pruning takes place. This shows that the lower bounds obtained by simply taking the optimal value of the parent problem are usually too weak to result in any pruning during the tree search. This can result in bad performances like the one observed for BR1 with AFIRO2. However, as a branching rule it appears to be most effective on binary problems such as AVGAS and QAP (6,4). As expected, pruning does work for BR2 and

Problem	n_i	n_c	m_e	m_i	Comments
Beale	3	0	0	1	Beale's problem, $0 \leq x_I \leq 5$
HS76	4	0	0	3	[38], No. 76, $0 \leq x_I \leq 5$
AVGAS (a), (b)	8	0	0	10	$0 \leq x_I \leq 1$, with different Hessians
AFIRO1 (a) – (c)	19	13	8	19	$0 \leq x_I \leq 5$, with different Hessians
AFIRO2 (a) – (c)	26	6	8	19	$0 \leq x_I \leq 5$, with different Hessians
QAP (6,4)	24	0	4	6	small quadratic assignment problem
RANDOM 1	10	0	0	12	5 random problems, $0 \leq x_I \leq 5$
RANDOM 2	20	0	0	20	5 random problems, $0 \leq x_I \leq 5$
RANDOM 3	20	10	0	30	5 random problems, $0 \leq x_I \leq 5$

Table 3.1: Description of MIQP Test Problems

Problem	BR1		BR2		BR3	
	generated	solved	generated	solved	generated	solved
Beale	7	7	7	4	7	4
HS76	5	5	5	3	5	3
AVGAS (a)	21	20	23	15	21	13
AVGAS (b)	19	19	27	16	19	13
AFIRO1 (a)	9	9	11	6	9	6
AFIRO1 (b)	17	17	17	10	17	14
AFIRO1 (c)	29	29	21	13	35	22
AFIRO2 (a)	4,867	4,867	393	221	5,565	4,010
AFIRO2 (b)	2,137	1,644	699	337	1,777	1,260
AFIRO2 (c)	not solved with stacksize 10,000					
QAP (6,4)	319	319	405	349	319	295
RANDOM 1	27.8	26.6	21.0	18.4	25.8	23.6
RANDOM 2	940	929	512	496	937	909
RANDOM 3	705	665	690	575	702	617

Table 3.2: Outcome of Tests: Number of generated and solved QP problems

apart from QAP (6,4) about 60% of the generated nodes are pruned due to the improved lower bounds. However, especially for the binary problems BR2 generates more problems than BR1 or BR3. BR3 finally seems to combine the best of both rules by adding the improved lower bounding feature to BR1. The largest improvement due to lower bounds is obtained for AFIRO2. This is probably due to the fact that these problems feature a large gap between the relaxed continuous optimum and the integer optimum and have a large range of integer variables.

Comparing the CPU times given in Table 3.3 for the three branching rules shows that the computation of the lower bounds is about as expensive as a single QP solve using the hot start facility in bqpd, the underlying QP solver. This implies that to base the branching rule on computing lower bounds for all possible branching decisions is very expensive and only worthwhile, if it results in considerable savings on the total number of problems solved. AFIRO2 is such a case and here BR2 is almost an order of magnitude faster than BR1 or BR3. Overall BR3 clearly beats BR1 and, apart from AFIRO2, also BR2 in terms of CPU time.

The results seem to indicate that BR2 is the best choice whenever there is a large gap between the continuous and the integer optimum, and this is supported by the fact that BR2 comes out best for the two pure integer random problems, but does not do quite as well for the mixed random

Problem	CPU times		
	BR1	BR2	BR3
Beale	0.03	0.02	0.01
HS76	0.02	0.02	0.01
AVGAS (a)	0.19	0.20	0.15
AVGAS (b)	0.17	0.21	0.15
AFIRO1 (a)	0.64	0.69	0.48
AFIRO1 (b)	0.91	0.82	0.77
AFIRO1 (c)	2.00	1.32	1.39
AFIRO2 (a)	215.55	22.17	175.43
AFIRO2 (b)	69.13	33.36	51.34
AFIRO2 (c)	not solved		
QAP (6,4)	9.75	16.42	10.05
RANDOM 1	0.63	0.56	0.58
RANDOM 2	111.34	98.66	110.65
RANDOM 3	288.14	290.94	271.19

Table 3.3: Outcome of Tests: CPU times to solve the MIQP

problem. Lower bounds can also strengthen existing branching rules such as BR1 and give a very competitive hybrid, BR3. Problems, however, that are known to have a large gap between continuous and integer optimum and a large range of integer variables are best solved by BR2.

Chapter 4

MILP Reformulation of P

4.1 Introduction

In this chapter and the next, a class of methods for MINLP problems is discussed, which provide an alternative to nonlinear branch–and–bound. These algorithms are based on the concept of defining an MILP *master problem*. Relaxations of such a master problem are then used in constructing algorithms for solving the MINLP problem.

There exist two related strategies for defining a master problem. Outer Approximation, proposed by Duran and Grossmann [15] and generalized by Yuan et. al. [79], and Benders Decomposition, introduced by Benders [6] and generalized by Geoffrion [26] and Flippo et. al. [20]. Both strategies obtain an MILP master program that is implicitly defined through the optimal solutions of NLP subproblems obtained by fixing the integer variables y in P. In this chapter these ideas leading to the statement of a master problem are clarified and extended.

The implicit nature of the MILP master problem makes its direct solution impracticable. It is therefore necessary to employ an iterative solution strategy based upon relaxations of the MILP master programs and this strategy is described in Chapter 5. These methods make use of any available software that exists for solving MILP problems. The approach is successful since it replaces the nonlinear by a linear (or quadratic) tree search, resulting in the solution of fewer NLP problems.

Outer Approximation and Benders Decomposition differ mainly in the way the master programs are derived. Both apply a projection onto the integer variables, but while the former makes use of supporting hyperplanes to represent P, the latter employs nonlinear duality theory. An important consequence of this difference is that Benders Decomposition adds only one constraint for every subproblem to the master program, while Outer Approximation adds one linearization for each constraint and the objective function for every subproblem. Moreover, the dualization implies that the continuous variables x are not present in the Benders master program. On the other hand it is not difficult to prove that Outer Approximation provides stronger cuts than Benders Decomposition and it has been observed in practice that Outer Approximation is usually superior to Benders Decomposition [15].

Section 4.2 presents the reformulation of P as an MILP master program. In order to improve the readability of the material, this is first done under the simplifying assumption that all integer assignments $y \in Y$ are feasible. Next a rigorous treatment of infeasible subproblems is presented, correcting an inaccuracy which occurs in [15] and [79]. Both Duran and Grossmann and Yuan et. al. give MILP master problems which are not correctly equivalent to P, with the serious consequence that their algorithms might cycle. It is reviewed how NLP–solvers detect infeasibility and a general framework is developed which is applicable to most common phase I approaches for resolving the

problems caused by infeasible subproblems. The final master program is then derived at the end of the section.

The second improvement over [15] and [79] is the simplicity of the proof which avoids the complication of semi–infinite programming problems. Moreover, the reformulation which is presented in Section 4.2 affords new insight into Outer Approximation. It can be seen, for example, that it suffices to add the linearizations of *strongly active* constraints to the master program. This is very important since it reduces the size of the MILP master program relaxations that are solved in the Outer Approximation Algorithms of Chapter 5.

Finally Section 4.3 presents the reformulation necessary to obtain the Benders master program. The Benders cuts that are obtained can be interpreted as supporting hyperplanes of the value function and a representation of the cuts in terms of objective and constraint gradients is given. This representation enables a comparison to be given with the corresponding Outer Approximations and the latter are shown to be stronger, generalizing an earlier result by Duran and Grossmann.

4.2 Outer Approximation of P

In this section the MINLP model problem P is reformulated as an MILP problem using Outer Approximation. The reformulation employs projection onto the integer variables and linearization of the resulting NLP subproblems by means of supporting hyperplanes. The convexity assumption allows an MILP formulation to be given where all supporting hyperplanes are collected in a single MILP.

The section is divided into three subsections. In the first a reformulation is presented under the simplifying assumption that all $y \in Y$ are feasible. It is indicated that a simplistic treatment of infeasible $y \in Y$ can lead to an incorrect MILP formulation. Consequently, the second part reviews how NLP solvers detect infeasibility and it is shown how this can lead to a rigorous treatment of infeasible $y \in Y$. Finally, in the last part developments of the two previous parts are combined and the correctly reformulated MILP master program is presented.

This section also simplifies the reformulation of [15] and [79], avoiding the complication of infinite programming problems. The new reformulation also shows that it is sufficient to add only the linearizations of strongly active constraints to the master program.

4.2.1 When all $y \in Y$ are feasible

In this subsection the simplifying assumption is made that all $y \in Y$ are feasible. This simplifies the reformulation of P greatly. At the end of the subsection the difficulties that arise if this assumption is dropped are briefly considered to motivate the next subsection that contains a rigorous treatment of infeasible $y \in Y$.

The first step in reformulating P is to define the NLP subproblem

$$\text{NLP}(y^j) \left\{ \begin{array}{ll} \min\limits_{x} & f(x, y^j) \\ \text{subject to} & g(x, y^j) \leq 0 \\ & x \in X \end{array} \right.$$

in which the integer variables are fixed at the value $y = y^j$. By defining $v(y^j)$ as the optimal value of the subproblem NLP(y^j) it is possible to express P in terms of a projection on to the y variables, that is

$$\text{proj(P)} \left\{ \min_{y^j \in Y} \{v(y^j)\}. \right. \tag{4.1}$$

The assumption that all $y \in Y$ are feasible implies that *all* subproblems are feasible. Let x^j denote an optimal solution of NLP(y^j) for $y^j \in Y$ (existence of x^j follows by the compactness of

X). Because a constraint qualification (assumption **A3**) holds at the solution of every subproblem NLP(y^j) for every $y^j \in Y$, it follows that proj(P) has the same optimal value as the problem

$$\min_{y^j \in Y} \left\{ \begin{array}{ll} \min\limits_{x} & f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ 0 \end{pmatrix} \\ \text{subject to} & 0 \geq g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ 0 \end{pmatrix} \\ & x \in X \end{array} \right\}.$$

In fact it suffices to include those linearizations of constraints about (x^j, y^j) which are strongly active at the solution of the corresponding subproblem. This is important since it implies that fewer constraints will have to be added to the master program in Chapter 5.

It is convenient to introduce a dummy variable $\eta \in \mathbb{R}$ into this problem, giving rise to the equivalent problem

$$\min_{y^j \in Y} \left\{ \begin{array}{ll} \min\limits_{x,\eta} & \eta \\ \text{subject to} & \eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ 0 \end{pmatrix} \\ & 0 \geq g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ 0 \end{pmatrix} \\ & x \in X \end{array} \right\}.$$

The convexity assumption **A1** implies that (x^i, y^i) is feasible in the inner optimization problem above for all $y^i \in Y$, where x^i is an optimal solution to NLP(y^i). Thus an equivalent MILP problem

$$\mathrm{M}_Y \left\{ \begin{array}{lll} \min\limits_{x,y,\eta} & \eta & \\ \text{subject to} & \eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} & \\ & & \forall y^j \in Y \\ & 0 \geq g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} & \\ & x \in X,\ y \in Y \text{ integer} & \end{array} \right.$$

is obtained. That is M_Y has one set of linearizations of the objective and constraint functions per integer point $y^j \in Y$.

Usually, not all $y \in Y$ give rise to feasible subproblems. Defining the sets

$$T = \{j : x^j \text{ is an optimal solution to NLP}(y^j)\}$$

$$\mathrm{V} = \{y \in Y : \exists\, x \in X \text{ with } g(x, y) \leq 0\}.$$

Then V is the set of all integer assignments y that give rise to feasible NLP–subproblems and T is the set of indices of these integer variables. Then P can be expressed as a projection on to the integer variables.

$$\text{proj(P)} \left\{ \min_{y^j \in V} \{v(y^j)\}. \right.$$

In this projection the set V replaces Y in (4.1). The equivalent MILP problem is now given by

$$\mathrm{M}_V \begin{cases} \min\limits_{x,y,\eta} & \eta \\ \text{subject to} & \eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} \\ & \qquad\qquad\qquad\qquad \forall j \in T \\ & 0 \geq g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} \\ & x \in X,\ y \in V \text{ integer} \end{cases}$$

obtained from M_Y by replacing Y by V.

It remains to find a suitable representation of the constraint $y \in V$ by means of supporting hyperplanes. The master problem given in [15] is obtained from problem M_V by replacing $y \in V$ by $y \in Y$. Duran and Grossmann justify this step by arguing that a representation of the constraints $y \in V$ is included in the linearizations in problem M_V. This argument is erroneous as the following example indicates. Consider the problem

$$\mathrm{P} \begin{cases} \min\limits_{x,y} & f(x,y) = -2y - x \\ \text{subject to} & 0 \geq x^2 + y \\ & y \in \{-1, 1\} \end{cases}$$

with solution $(x^*, y^*) = (1, -1)$ and $f^* = 1$. The master program given in [15] can be written as

$$\mathrm{M}_{DG} \begin{cases} \min\limits_{x,y} & f(x,y) = -2y - x \\ \text{subject to} & 0 \geq 1 + 2(x-1) + y \\ & y \in \{-1, 1\} \end{cases}$$

and problem M_{DG} has the solution $(x^*, y^*) = (0, 1)$ and $f^* = -2$, which is infeasible in P. Thus the value $y = 1$ is not excluded in M_{DG} and hence M_{DG} and P are not equivalent in this case.

This small example clearly illustrates that it is necessary to include information from infeasible subproblems. However care has to be taken when choosing the value of x about which to linearize the subproblem, as is illustrated by choosing $x = \frac{1}{2}$ for the infeasible subproblem $y = 1$. This choice results in the constraint

$$\tfrac{1}{4} + (x - \tfrac{1}{2}) + y \leq 0$$

being added to M_V, which does not exclude $y = 1$ from M_V.

In order to derive cuts that correctly exclude infeasible $y \in Y$ from the master program it is necessary to examine how NLP solvers detect infeasibility and this is done in the next subsection.

4.2.2 Infeasibility in NLP problems

This subsection provides a framework in which most common methods for detecting infeasibility in NLP problems can be included. NLP solvers detect infeasibility in an NLP problem in many ways. Consider the constraints of an NLP problem.

$$\begin{cases} g_i(x) \leq 0, & i = 1, 2, \ldots, m \\ x \in X \subset \mathbb{R}^n \end{cases}$$

where the set X might for example be simple upper and lower bounds on x. Two obvious possibilities for finding a feasible point are to attempt to minimize an l_1 or l_∞ sum of constraint violations, that is

$$\min_x \sum_{i=1}^{m} g_i^+(x),\ x \in X$$

or

$$\min_x \max_{i=1,\ldots,m} g_i^+(x),\ x \in X$$

where $a^+ = \max(a, 0)$. Other methods aim to maintain feasibility in any constraint residual once it has been established. For example an l_1 problem of the form

$$\begin{cases} \min\limits_x & \sum\limits_{i \in J^\perp} g_i^+(x) \\ \text{subject to} & g_j(x) \le 0,\ j \in J \\ & x \in X \end{cases}$$

might be solved, in which the set J is the set of constraints which are currently feasible in the algorithm and $J^\perp$ is its complement. Alternatively constraints may be driven to feasibility one at a time, whilst maintaining feasibility for constraints indexed by $j \in J$. In this case a problem

$$\begin{cases} \min\limits_x & g_i^+(x) \\ \text{subject to} & g_j(x) \le 0,\ j \in J \\ & x \in X \end{cases}$$

is being solved at any one time.

In all these cases, if the constraints are inconsistent, then the phase I approach terminates at a point, x' say, with an indication that the problem is infeasible. At this stage it is possible to write down an equivalent *feasibility problem* F that has been solved, in which there are weights that depend on the type of phase I approach. The weights w_i are nonnegative and are not all zero. Infeasibility in the NLP problem is then equivalent to having obtained a solution x' of F for which the objective function is greater than zero.

$$\mathrm{F} \begin{cases} \min\limits_x & \sum\limits_{i \in J^\perp} w_i g_i^+(x) \\ \text{subject to} & g_j(x) \le 0,\ j \in J \\ & x \in X. \end{cases}$$

All of the above cases fit into this framework. (In the l_∞ case, there exist nonnegative weights at the solution such that $\sum w_i = 1$ and $w_i = 0$ if g_i does not attain the maximum value.)

In the context of the problem NLP(y^k) introduced in the previous subsection, the general feasibility problem has the form

$$\mathrm{F}(y^k) \begin{cases} \min\limits_x & \sum\limits_{i \in J^\perp} w_i^k g_i^+(x, y^k) \\ \text{subject to} & g_j(x, y^k) \le 0,\ j \in J \\ & x \in X \end{cases}$$

where y^k is some fixed assignment of the integer variables y of problem P and the nonnegative weights w_i^k may differ for different y^k . An important property of F(y^k) is expressed in the following lemma.

Lemma 4.2.1 *If NLP(y^k) is infeasible, so that x^k solves F(y^k) with*

$$\sum_{i \in J^\perp} w_i^k (g_i^k)^+ > 0 \tag{4.2}$$

then $y = y^k$ is infeasible in the constraints

$$\begin{aligned} 0 &\geq g_i^k + (\nabla g_i^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \quad \forall i \in J^\perp \\ 0 &\geq g_j^k + (\nabla g_j^k)^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \quad \forall j \in J, \end{aligned}$$

for all $x \in X$.

Proof:
Assume that y^k is feasible in the above constraints, so that there exists an $\hat{x} \in X$ such that $(\hat{x}, y^k)$ satisfies the following set of inequalities:

$$0 \geq g_i^k + (\nabla g_i^k)^T \begin{pmatrix} \hat{x} - x^k \\ 0 \end{pmatrix} \forall i \in J^\perp \tag{4.3}$$

$$0 \geq g_j^k + (\nabla g_j^k)^T \begin{pmatrix} \hat{x} - x^k \\ 0 \end{pmatrix} \forall j \in J. \tag{4.4}$$

It is convenient to handle the condition $\hat{x} \in X$ by introducing the normal cone $\partial X(x^k)$ at x^k. The normal cone $\partial X(x^k)$ is defined as the set of vectors u that satisfy the inequality

$$u^T(x - x^k) \leq 0 \qquad\qquad \forall x \in X. \tag{4.5}$$

The first order Kuhn–Tucker conditions for the feasibility problem F(y^k) imply that there exist multipliers $\lambda_j \geq 0,\ j \in J$ and a vector $u \in \partial X(x^k)$ such that

$$\lambda_j g_j^k = 0,\ \forall j \in J \tag{4.6}$$

$$\sum_{i \in J^\perp} w_i^k \nabla g_i^k + \sum_{j \in J} \lambda_j \nabla g_j^k + u = 0. \tag{4.7}$$

After multiplying (4.3) by $w_i^k \geq 0$ and (4.4) by $\lambda_j \geq 0$, summing, and adding to (4.5) the following inequality is obtained.

$$0 \geq \sum_{i \in J^\perp} w_i g_i^k + \sum_{j \in J} \lambda_j g_j^k + [\ \sum_{i \in J^\perp} w_i \nabla g_i^k + \sum_{j \in J} \lambda_j \nabla g_j^k + \begin{pmatrix} u \\ 0 \end{pmatrix}]^T \begin{pmatrix} \hat{x} - x^k \\ 0 \end{pmatrix}. \tag{4.8}$$

Substituting (4.6) and (4.7) the inequality (4.8) becomes

$$0 \geq \sum_{i \in J^\perp} w_i^k g_i^k$$

which contradicts (4.2) and proves the lemma.
□

The proof of Lemma 4.2.1 also shows that it suffices to consider linearizations of constraints which are *strongly active* (i.e. whose multipliers are nonzero). This implies that it is sufficient to add linearizations of strongly active constraints to the master program relaxations of Chapter 5.

It is possible to derive a similar lemma for the NLP(y^k) subproblems in the context of Benders Decomposition. Since Benders Decomposition employs nonlinear duality theory it is more convenient to write the subproblem as

$$\mathrm{F}'(y^k)\begin{cases} \min\limits_{x,y} & \sum\limits_{i\in J^\perp} w_i^k g_i^+(x,y) \\ \text{subject to} & g_j(x,y)\le 0,\ j\in J \\ & y = y^k \\ & x\in X,\ y\in Y. \end{cases}$$

An equivalent statement to Lemma 4.2.1 can now be made.

Lemma 4.2.2 *If NLP(y^k) is infeasible, so that x^k solves $F'(y^k)$ with*

$$\sum_{i\in J^\perp} w_i^k (g_i^k)^+ > 0 \tag{4.9}$$

then $y = y^k$ is infeasible in the constraint

$$0 \ge \sum_{i\in J^\perp} w_i^k (g_i^k)^+ + (\nu^k)^T (y^k - y) \tag{4.10}$$

where ν^k is the optimal multiplier of the constraint $y = y^k$ in $F'(y^k)$.
Moreover any y^l for which NLP(y^l) is feasible satisfies the cut (4.10).

Proof:
The first part of the Lemma follows by substituting $y = y^k$ in (4.10). The second part follows from the convexity of P. Let y^l be such that NLP(y^l) is feasible. Then it follows that

$$0 \ge g^l.$$

Multiplying each individual inequality by w_i^k and λ_j^k and summing gives

$$0 \ge \sum_{i\in J^\perp} w_i^k g_i^l + \sum_{j\in J} \lambda_j^k g_j^l.$$

The convexity of g implies that

$$0 \ge \sum_{i\in J^\perp} w_i^k g_i^k + \sum_{i\in J^\perp} w_i^k (\nabla g_i^k)^T (y^l - y^k) + \sum_{j\in J} \lambda_j^k g_j^k + \sum_{j\in J} \lambda_j^k (\nabla g_j^k)^T (y^l - y^k).$$

The first order necessary conditions for $\mathrm{F}'(y^k)$ imply that

$$-\nu^k = \sum_{i\in J^\perp} w_i^k (\nabla g_i^k) + \sum_{j\in J} \lambda_j^k (\nabla g_j^k)^T (y^l - y^k)$$

so that y^l satisfies the cut (4.10).
□

Both Lemma 4.2.1 and Lemma 4.2.2 play an important role in deriving a rigorous MILP formulation of P. The next subsection completes the derivation of the MILP master program through outer approximations. Section 4.3 shows how Benders' Decomposition can be derived in a similar fashion.

4.2.3 The general case

This subsection completes the derivation of the MILP master program by combining the developments of the previous two subsections. The small example presented in Subsection 4.2.1 clearly illustrates that it is necessary to ensure that integer assignments which produce infeasible subproblems are also infeasible in the master program M. Let the integer assignment y^k produce an infeasible subproblem and denote

$$S = \left\{ k : \text{NLP}(y^k) \text{ is infeasible and } x^k \text{ solves F}(y^k) \right\}.$$

Note that S is the complement of the set T defined in Subsection 4.2.1. It then follows directly from Lemma 4.2.1 of Subsection 4.2.2 that the constraints

$$0 \geq g^k + [\nabla g^k]^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \ \forall k \in S$$

exclude all integer assignments y^k for which NLP(y^k) is infeasible. Thus a general way to correctly represent the constraints $y \in V$ in M_V is to add linearizations from F(y^k) when infeasible subproblems are obtained, giving rise to the following MILP master problem.

$$\text{M} \left\{ \begin{array}{lll} \min\limits_{x,y,\eta} & \eta & \\ \text{subject to} & \eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} & \\ & & \forall j \in T \\ & 0 \geq g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} & \\ & 0 \geq g^k + [\nabla g^k]^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} & \forall k \in S \\ & x \in X,\ y \in Y \text{ integer.} & \end{array} \right.$$

Continuing the example of Subsection 4.2.1 a suitable form of feasibility problem F is

$$\min_x (x^2 + 1)^+$$

which is solved by $x = 0$. Thus the constraint

$$y \leq 0$$

is added to M_{DG} which correctly excludes the infeasible integer assignment $y = 1$. The correctly reformulated master program is illustrated in Figure 4.1.

The development of the preceding two subsections provides a proof of the following result:

Theorem 4.2.1 *If assumptions* **A1**, **A2** *and* **A3** *hold, then M is equivalent to P in the sense that* (x^*, y^*) *solves P if and only if it solves M.*

Problem M is an MILP problem, but it is not practical to solve M directly, since this would require all subproblems NLP(y^j) to be solved first. This would be a very inefficient way of solving problem P. Another practical disadvantage of M is that it contains a very large number of constraints. For example if $Y = \{0, 1\}^p$ and if P has m constraints then M would contain $2^p \cdot (m + 1)$ constraints. Therefore, instead of attempting to solve M directly, relaxations of M are used in an iterative process that is the subject of the next chapter.

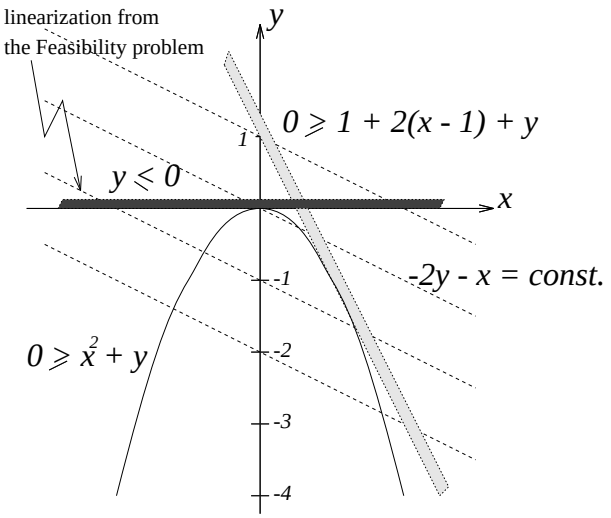

Figure 4.1: Correct master program for the small example

4.3 Benders Decomposition for P

Benders Decomposition was first suggested by Benders [6]. Geoffrion [26] generalized it to cover a wider range of problems and Flippo et. al. [20] proved many new results in a more general setting. Like Outer Approximation, the MINLP model problem P is reformulated as an MILP master program. However, instead of using supporting hyperplanes, Benders Decomposition makes use of nonlinear duality theory to derive this equivalent formulation.

This section briefly presents the reformulation of P as an equivalent MILP master program. The main tools required here are projection onto the integer variables and dualization of the resulting NLP subproblems. The advantage of this reformulation lies in the fact that it does not contain the continuous variables x and that only one constraint is added per subproblem. The disadvantage, however, is that the cuts derived from Benders Decomposition are weaker than the corresponding linearizations derived by Outer Approximation.

It is possible to interpret the Benders cuts as supporting hyperplanes of the value function. Benders Decomposition can therefore be interpreted as the Outer Approximation of a nonsmooth convex MINLP as consider in Chapter 7.

The derivation of an equivalent MILP master program follows the very instructive derivation of Flippo et. al. [20] for more general problems. Their reformulation is directly applicable to P and there is no need for Geoffrion's "Property $\mathcal{P}$", described towards the end of this section, since it is implied by the stronger convexity assumption. As in Outer Approximation the first step is to express P in terms of a projection onto the integer variables.

$$\text{proj(P)} \left\{ \min_{y^j \in V} \{v'(y^j)\}. \right.$$

where V is the set of all integer assignments that give rise to feasible NLP subproblems as defined

in Section 4.2 and v' is the optimal value of the NLP subproblem defined by

$$\text{NLP}'(y^j) \begin{cases} \min\limits_{x,y} & f(x,y) \\ \text{subject to} & g(x,y) \leq 0 \\ & y = y^j \\ & x \in X,\ y \in conv(Y). \end{cases}$$

The use of the copying constraint $y = y^j$ turns out to be crucial in the derivation of the dual of NLP$'(y^j)$, since it implies that the dual feasible space is independent of y^j. If the dual feasible space were not independent of y^i then the master program which is derived in this chapter would be computationally intractable. In order to escape this difficulty, Geoffrion [26] introduces a so called "Property $\mathcal{P}$" which ensures that the Benders cuts can be derived with little additional effort independent of y^i (see also [22]).

Instead of representing NLP$'(y^j)$ by its supporting hyperplanes as in Outer Approximation, the Lagrangian dual of NLP$'(y^j)$

$$\text{DNLP}'(y^j) \quad \left\{ \max_{\lambda \geq 0, \mu} \left(\inf_{x \in X, y \in conv(Y)} [f(x,y) + \lambda^T g(x,y) + \mu^T (y - y^j)] \right) \right.$$

is used. Since a constraint qualification holds at the solution of NLP$'(y^j)$ and since it is a convex programming problem, its dual DNLP$'(y^j)$ is a strong dual. Thus the optimal value of primal and dual are equal and any optimal solution (λ^j, μ^j) of the dual characterizes the set of optimal solutions of the primal as the minimum of the Lagrangian over $X \times conv(Y)$ which also satisfy primal feasibility and complementarity (e.g. [25], Theorem 3). The projected problem proj(P) is now equivalent to

$$\min_{y^j \in V} \left\{ \begin{array}{ll} \min\limits_{\eta} & \eta \\ \text{subject to} & \eta \geq \max\limits_{\lambda \geq 0, \mu} \left(\inf\limits_{x \in X, y \in conv(Y)} [f(x,y) + \lambda^T g(x,y) + \mu^T (y - y^j)] \right) \end{array} \right.$$

where the dummy variable η has been introduced. By using the equivalence

$$\eta \geq \max_i r_i \ \Leftrightarrow \ \eta \geq r_i \ \forall i$$

it is possible to express proj(P) as

$$\min_{y^j \in V} \left\{ \begin{array}{ll} \min\limits_{\eta} & \eta \\ \text{subject to} & \eta \geq \inf\limits_{x \in X, y \in conv(Y)} [f(x,y) + \lambda^T g(x,y) + \mu^T (y - y^j)] \ \forall \lambda \geq 0,\ \forall \mu \end{array} \right.$$

Since there exists only a finite number of primal optimal solutions (x^i, y^i), each with corresponding dual solution (λ^i, μ^i) it is possible to rewrite the inner optimization in the above problem as

$$\min_{y^j \in V} \left\{ \begin{array}{ll} \min\limits_{\eta} & \eta \\ \text{subject to} & \eta \geq f^i + (\lambda^i)^T g^i + (\mu^i)^T (y^i - y^j) \end{array} \right.$$

Since P is convex any pair (η^j, y^j) (where η^j denotes the optimal value of the NLP$'(y^j)$–subproblem) is feasible in the inner optimization and problem P can be written as

$$\left\{ \begin{array}{lll} \min\limits_{\eta, y} & \eta & \\ \text{subject to} & \eta \geq f^i + (\lambda^i)^T g^i + (\mu^i)^T (y^i - y) & \forall y^i \in V \\ & y \in V. & \end{array} \right.$$

Noting that $(\lambda^j)^T g^j = 0$ and utilizing a suitable dual representation of the constraint $y \in V$, the MILP master program

$$\mathrm{M}_{BD} \left\{ \begin{array}{lll} \min\limits_{y,\eta} & \eta & \\ \text{subject to} & \eta \geq f^j + (\mu^j)^T (y^j - y) & \forall j \in T \\ & 0 \geq \sum\limits_{i \in J^\perp} w_i^k (g_i^k)^+ + (\nu^k)^T (y^k - y) & \forall k \in S \\ & y \in Y \text{ integer.} & \end{array} \right.$$

is obtained, where the sets S and T are those defined in Section 4.2.

It is noteworthy to remark that the master program M_{BD} does not contain the continuous variables x and contains only one constraint per NLP subproblem. Although this seems to be advantageous compared to Outer Approximation, practical experience shows Outer Approximation to be computationally faster [15].

An interesting interpretation of the Benders cut can be obtained by using a standard interpretation of Lagrange multipliers. Consider the *value function* $v'(y^j)$ defined by

$$v'(y^j) = \left\{ \begin{array}{lll} \min\limits_{x,y} & f(x,y) & \\ \text{subject to} & g(x,y) \leq 0 & \\ & y = y^j & \\ & x \in X,\ y \in Y. & \end{array} \right.$$

for $y^j \in Y$. The Lagrange multiplier μ^j of the *copy constraint* $y = y^j$ gives the rate of change in the objective function upon changes in the constraints. Hence it is possible to interpret the Benders cut

$$\eta \geq f^j + (\mu^j)^T (y - y^j)$$

as a first order approximation to the value function $v(y^j)$ about y^j. Geoffrion [25] shows that μ^j is in fact a subgradient of $v(y^j)$.

It is possible to write the Benders cuts entirely in terms of objective and constraint gradients, if f and g are continuously differentiable. The Kuhn–Tucker necessary condition for $\mathrm{NLP}'(y^j)$ and $\mathrm{F}'(y^k)$ imply that

$$\begin{aligned} 0 &= \nabla_y f^j + [\nabla_y g^j] \lambda^j + \mu^j \\ 0 &= \sum_{i \in J^\perp} w_i^k \nabla_y g_i^k + \sum_{j \in J} \lambda_j^k \nabla_y g_j^k + \nu^k. \end{aligned}$$

Solving both first order conditions for μ^j and ν^k respectively gives the Benders cuts in terms of the objective and constraint gradients only.

These expressions for μ^j and ν^k also afford insight into the relationship between Outer Approximation and Benders Decomposition. Consider the Outer Approximations for a *feasible* NLP subproblem

$$\eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} \tag{4.11}$$

$$0 \geq g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix}. \tag{4.12}$$

Premultiplying (4.12) by λ^j, summing and adding to (4.11) gives the valid cut

$$\eta \geq f^j + (\lambda^j)^T g^j + (\nabla f^j + [\nabla g^j] \lambda^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix}. \tag{4.13}$$

Substituting for the Kuhn–Tucker conditions

$$\begin{aligned} 0 &= (\lambda^j)^T g^j \\ 0 &= \nabla_x f^j + [\nabla_x g^j]\lambda^j \end{aligned}$$

in the valid cut (4.13) the Benders cut

$$\eta \geq f^j + (\nabla_y f^j + [\nabla_y g^j]\lambda^j)^T (y - y^j)$$

is obtained.

This development shows that Outer Approximation provides stronger cuts than Benders Decomposition, so that in general Outer Approximation will provide a stronger master program. This has also been observed in practice, where Outer approximation usually outperforms Benders Decomposition [15].

Finally, an important difference between Geoffrion's and Flippo et. al.'s derivation of Benders' Decomposition is that the latter does not require a so called *Property* $\mathcal{P}$ to hold. Property $\mathcal{P}$ requires that the infimum in the Benders cut

$$\inf_{x \in X} \{f(x, y) + \lambda^T g(x, y)\}$$

can be taken essentially independent of y. Flippo et. al. show that Property $\mathcal{P}$ is implied by the stronger convexity assumption **A1**. However, in the absence of the stronger convexity assumption Property $\mathcal{P}$ enables Benders' Decomposition to treat certain nonconvex problems that are not readily solved by other methods such as branch–and–bound and outer approximation. Geoffrion [26] applies Benders' Decomposition to the variable factor programming problem, which is non–convex but for which Property $\mathcal{P}$ holds.

Chapter 5

The Algorithms

5.1 Introduction

It has been shown in Chapter 4 how the MINLP model problem P can be reformulated as an MILP problem. However, this reformulation requires the solution of *all* NLP–subproblems, so that the setting up of M is equivalent to the solution of P. Duran and Grossmann [15] and more recently Quesada and Grossmann [60] have suggested two algorithms based upon solving relaxations of the master program M in an iterative procedure. This chapter generalizes both algorithms and examines their worst case behaviour. This chapter also shows how the ideas of outer approximation can be employed to devise a new algorithm based on Lagrangean Decomposition (e.g. [54]).

In Section 5.2 a new *linear outer approximation* algorithm is developed, based on solving successive MILP relaxations of the master problem M of Chapter 4, and it is proved that the algorithm terminates finitely. The algorithm owes much to the innovative work of Duran and Grossmann [15] in developing an outer approximation algorithm, but the new algorithm improves on this work in a number of ways. The problem formulation allows the integer variables y to occur nonlinearly in f and g, similar to the outer approximation algorithm of Yuan et. al. [79]. Improving on both [15] and [79], a new and more simple proof of termination is given. The occurrence of infeasible solutions to NLP subproblems is treated in a rigorous way which is generally applicable to many different methods for solving Phase I problems. The resulting method is also suitable for pure INLP problems in which the x variables in P are absent, which is not the case for the Duran and Grossmann formulation.

The practical performance of the resulting algorithm has proved to be similar to that of the Duran and Grossmann algorithm in cases where the latter is applicable. However, the worst case performance of the algorithm is studied and an example is provided which shows that the algorithm can be very inefficient. This subsequently motivates the investigation of a *quadratic outer approximation* algorithm which solves MIQP master problems in an attempt to take second order information into account.

The outer approximation algorithm has been implemented as DICOPT by Kocis and Grossmann [43]. It has proved very successful in practice and Duran and Grossmann show in a small computational study [15] that it is superior to both nonlinear branch–and–bound and to Benders Decomposition. The outer approximation algorithms presented in this chapter have also been implemented and practical experience with these algorithms is presented in Chapter 6. The experience gained with the present implementations indicate that the performance of the MINLP algorithms depends on the sub–class of MINLP problem that is being solved.

In Section 5.3 a new approach to the solution of the master program relaxations due to Quesada and Grossmann [60] is discussed. It is termed *LP/NLP based branch and bound algorithm*. This

algorithm avoids resolving successive relaxations of the MILP master program M and instead incorporates the NLP–subproblems into the branch and bound search, updating all pending nodes of the tree as new cuts are added from NLP subproblems. As for linear outer approximation, it is possible to show that this algorithm can perform very poorly, if curvature information plays a role in the problem. Using similar ideas to those developed for a quadratic outer approximation algorithm, it is possible to include curvature information into this process giving rise to a new *QP/NLP based branch and bound algorithm.*

Finally Section 5.5 generalizes a Lagrangean Decomposition Algorithm proposed by Michelon and Maculan [54] for linearly constrained integer problems. The algorithm employs outer approximations similar to those used in the outer approximation algorithms of Sections 5.2 and 5.3. Considerations of the bounds generated by Lagrangean Decomposition indicate that the new algorithm is likely to be less efficient than outer approximation.

5.2 The Outer–Approximation Algorithms

The outer approximation algorithm of Duran and Grossman [15] has been used to solve a number of practical optimization problems related to process engineering (e.g. [42], [43], [73]). It has been found to be an efficient alternative to nonlinear branch–and–bound and Benders Decomposition for this class of problems. In this section the outer approximation algorithms of Duran and Grossmann and of Quesada and Grossmann [60] are generalized to a wider class of problems.

This section describes, how relaxations of the master program M, developed in Section 4.2 can be employed to solve the model problem P. The resulting algorithm is termed *linear outer approximation*. It is shown to iterate finitely between NLP subproblems and MILP master program relaxations. This algorithm is seen to be less efficient if curvature information is present in the problem. A worst case example, in which linear outer approximation visits all integer assignments is derived that motivates the introduction of a second order term into the MILP master program relaxations.

Each iteration of the linear outer approximation algorithm chooses a new integer assignment y^i and attempts to solve NLP(y^i). Either a feasible solution x^i is obtained or infeasibility is detected and x^i is the solution of a feasibility problem F(y^i) (other pathological cases are eliminated by the assumption that the set X is compact). The algorithm replaces the sets T and S in M by the sets

$$\begin{aligned} T^i &= \{j \mid j \leq i : x^j \text{ is an optimal solution to NLP}(y^j)\} \\ S^i &= \left\{k \mid k \leq i : \text{NLP}(y^k) \text{ is infeasible and } x^k \text{ solves } F(y^k)\right\}. \end{aligned}$$

It is also necessary to prevent any $y^j,\ j \in T^i$ from becoming the solution of the relaxed master problem. This can be done by including a constraint

$$\eta < \text{UBD}^i$$

where

$$\text{UBD}^i = \min\{f^j : \ \ j \in T^i\}.$$

Thus the following master problem is defined

$$
M^i \left\{ \begin{array}{lll}
\min\limits_{x,y,\eta} & \eta & \\
\text{s. t.} & \eta < \text{UBD}^i & \\
& \eta \geq f^j + \nabla(f^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} & \\
& & \forall j \in T^i \\
& 0 \geq g^j + \nabla[g^j]^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} & \\
& 0 \geq g^k + \nabla[g^k]^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} & \forall k \in S^i \\
& x \in X,\ y \in Y \text{ integer.} &
\end{array} \right.
$$

The algorithm solves M^i to obtain a new integer assignment y^{i+1}, and the whole process is repeated iteratively. Figure 5.1 illustrates the different stages of the algorithm.

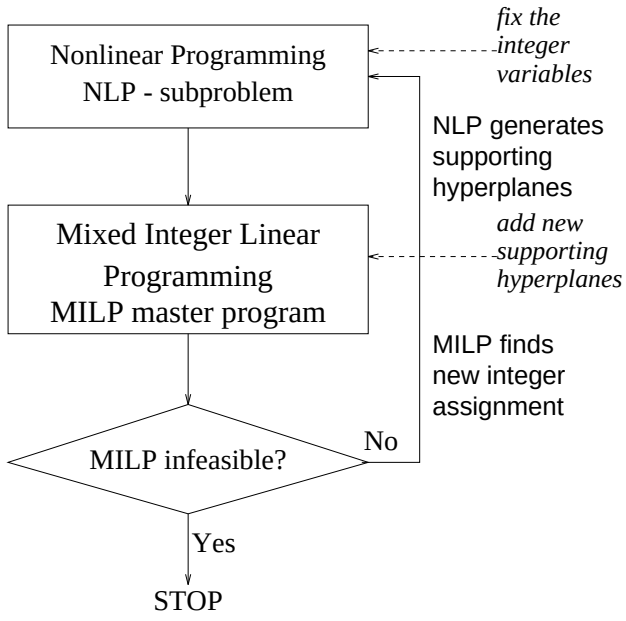

Figure 5.1: Illustration of Algorithm 1

A detailed description of the algorithm is as follows.

Algorithm 1: Linear Outer Approximation

Initialization: y^0 is given; set $i = 0$, $T^{-1} = \emptyset$, $S^{-1} = \emptyset$ and $\text{UBD}^{-1} = \infty$.

REPEAT

1. Solve the subproblem NLP(y^i), or a feasibility problem F(y^i) if NLP(y^i) is infeasible, and let the solution be x^i.

2. Linearize the objective and (active) constraint functions about (x^i, y^i). Set $T^i = T^{i-1} \cup \{i\}$ or $S^i = S^{i-1} \cup \{i\}$ as appropriate.

3. **IF** (NLP(y^i) is feasible and $f(x^i, y^i) < \text{UBD}^{i-1}$) **THEN**
   update current best point by setting $x^* = x^i$, $y^* = y^i$, $\text{UBD}^i = f(x^i, y^i)$.
   ELSE
   Set $\text{UBD}^i = \text{UBD}^{i-1}$.

4. Solve the current relaxation M^i of the master program M, giving a new integer assignment y^{i+1} to be tested in the algorithm. Set $i = i + 1$.

UNTIL (M^i is infeasible).

The algorithm also detects whether or not P is infeasible. If UBD$= \infty$ on exit then all integer assignments that are visited by the algorithm are infeasible (i.e. Step 3 is not invoked). The use of upper bounds on η and the definition of the sets T^i and S^i ensure that no y^i is replicated by the algorithm. This enables a proof to be given that the algorithm terminates after a finite number of steps, provided that there is only a finite number of integer assignments.

Theorem 5.2.1 *If assumptions* **A1**, **A2**, **A3** *and* **A4** *hold, and* $|Y| < \infty$, *then Algorithm 1 terminates in a finite number of steps at an optimal solution of P or with an indication that P is infeasible.*

Proof:
It suffices to show that no integer assignment is generated twice by the algorithm. The finiteness of the algorithm then follows from the finiteness of the set Y. Let $l \leq i$. If $l \in S^i$ it follows from Lemma 4.2.1 of Section 4.2.2 that the cuts introduced from the solution of the feasibility problem F(y^l) exclude y^l from any subsequent master program.
If $l \in T^i$ it is assumed that y^l is feasible in M^i and a contradiction is sought. Solving M^i gives the solution $(\eta^{i+1}, \hat{x}^{i+1}, y^l)$, which must satisfy the following set of inequalities:

$$\eta^{i+1} < \text{UBD}^i \leq f^l \tag{5.1}$$

$$\eta^{i+1} \geq f^l + (\nabla f^l)^T \begin{pmatrix} \hat{x}^{i+1} - x^l \\ 0 \end{pmatrix} \tag{5.2}$$

$$0 \geq g^l + (\nabla g^l)^T \begin{pmatrix} \hat{x}^{i+1} - x^l \\ 0 \end{pmatrix}. \tag{5.3}$$

Since x^l is the optimal solution to NLP(y^l) and a constraint qualification holds (**A3**), no feasible descent direction exists at x^l, that is

$$0 \geq g^l + (g^l)^T \begin{pmatrix} \hat{x}^{i+1} - x^l \\ 0 \end{pmatrix}$$

$$\Rightarrow (\nabla f^l)^T \begin{pmatrix} \hat{x}^{i+1} - x^l \\ 0 \end{pmatrix} \geq 0. \tag{5.4}$$

Substituting (5.4) into (5.2) gives

$$\eta^{i+1} \geq f^l$$

which contradicts (5.1). Thus y^l is infeasible for all $l \in S^i$ and $l \in T^i$.

Finally it is shown that Algorithm 1 always terminates at a solution of P or with an indication that P is infeasible. If P is feasible, then let an optimal solution to P be given by (x^*, y^*) with optimal value f^* (any other optimal solution has the same objective value and the algorithm does not distinguish between them). Since M is a relaxation of P, f^* is an upper bound on the optimal value of M, which is attained at (x^*, y^*). Now assume that the algorithm terminates with an indicated solution (x', y') with $f' > f^*$ (i.e. not optimal). Since $\text{UBD}^i = f' > f^*$ it follows that (x^*, y^*) must be feasible in the previous relaxation of M, which contradicts the assumption that the algorithm terminates at (x', y'). If on the other hand P is infeasible then all NLP(y^j)–subproblem are infeasible and the algorithm never updates the upper bound UBD, and hence exits with UBD$= \infty$ indicating an infeasible problem.
□

It has been indicated in Chapter 1 that Assumption **A3** is not merely a technical assumption. In fact, this assumption ensures that the linearizations cut out any integer assignment that has been visited by the algorithm. The following example illustrates the importance of a constraint qualification.

$$\begin{cases} \min\limits_{x,y} & y + (x-1)^2 \\ \text{subject to} & -y + x^2 + 1 \leq 0 \\ & y - 1 \leq 0 \\ & x \in [-5, 5],\ y \in \{0, 1, 2\} \end{cases}$$

For $y = 1$ the constraint qualification is *not* satisfied. Thus starting outer approximation at $y^0 = 1$, the solution to the NLP–subproblem is $x^0 = 0$ and the following master program is solved next.

$$\begin{cases} \min\limits_{\eta,x,y} & \eta \\ \text{subject to} & \eta < 2 \\ & \eta \geq y + 1 - 2x \\ & 0 \geq -y + 1 \\ & 0 \geq y - 1 \\ & x \in [-5, 5],\ y \in \{0, 1, 2\} \end{cases}$$

This problem is feasible and its optimal solution is $(\eta^1, y^1, x^1) = (-8, 1, 5)$ and the outer approximation routine cycles. The reason for this behaviour is that the absence of a constraint qualification means that there could be linearized feasible descent directions in x so that an integer assignment could be generated again.

It can be observed from the proof that it is not necessary to solve M^i for optimality in Algorithm 1, as long as a new integer assignment is obtained from M^i. However, if M^i is solved for optimality then the upper bound on η can be supplemented by a weak lower bound

$$\eta \geq \eta^i$$

where η^i is the solution value of M^{i-1}. This lower bound can improve the efficiency of the MILP solver by cutting out branches of the branch and bound tree that need not be examined. It is worth

remarking that the method of proof used here is much more simple (especially with respect to the derivation of the master problem M) than that of Duran and Grossmann which is based on integer polyhedra and linear programming theory.

There are a number of practical considerations that arise when implementing the algorithm. It is mentioned in Chapter 4 that it is worthwhile to include only those constraints that are active at a solution of the subproblem NLP(y^i) so that fewer linearizations are added to the master program at each iteration. If this is done it might not be necessary to include a constraint dropping procedure that scans the constraints of the master program to keep its size small. On the other hand, adding fewer constraints to the master program implies that the master program relaxations are weaker which could result in a larger number of iterations. The numerical experiments presented in Chapter 6 are obtained by adding all linearizations to the master program.

In practice the constraint

$$\eta < \text{UBD}^i$$

would not be used, but rather

$$\eta \leq \text{UBD}^i - \epsilon$$

where ϵ is some small user supplied accuracy. The strict upper bound $\eta < \text{UBD}^i$ only guarantees finite termination in the idealistic case where all solvers involved are exact. Chapter 8 indicates how ϵ should be chosen so that finite termination is guaranteed for inexact arithmetic. The algorithm can then only be guaranteed to provide an ϵ–optimal solution to P. If the value of the objective function is very small (of order ϵ), then it might be better to consider the relative rather than the absolute error. Currently the mixed error term

$$\eta \leq \text{UBD}^i - \epsilon(1 + |\text{UBD}^i|)$$

is preferred.

As mentioned in Chapter 4 the reformulation includes pure INLP problems and this makes Algorithm 1 applicable to pure INLP problems, in which case Step 1 of the algorithm (the inner optimization over the continuous variables) becomes redundant.

Practical experience with linear outer approximation given in [15] indicates that outer approximation is superior both to nonlinear branch and bound and Generalized Benders Decomposition, although the test problems are limited to the case where f and g are both linear functions in y. It is of interest to know whether this is always the case, or if there exist situations in which the outer approximation algorithm performs badly. With this in mind a worst case example has been devised in which Algorithm 1 visits all integer feasible points in the problem before finding the solution, even though the initial assignment y^0 is adjacent to the optimal assignment. The example is

$$\begin{cases} \min\limits_{y} & f(y) = (y - \epsilon)^2 \\ \text{s. t.} & y \in \{0, \epsilon, \ldots, \frac{1}{2}, 1\} \end{cases}$$

in which $\epsilon = 2^{-p}$ for some $p > 1$.

Starting with $y^0 = 0$, which is the adjacent value to the solution $y^* = \epsilon$, the next iterate is $y^1 = 1$, which is an extreme feasible point. Algorithm 1 then works its way back to the solution by visiting each remaining integer assignment $y^i = 2^{-i+1},\ i = 2, 3, \ldots, p+1$ in turn. Figure 5.2 illustrates the situation for $p = 3$ and the shaded boxes indicate the various supporting hyperplanes that are generated. This example is also a worst case example for Generalized Benders Decomposition but is solved by nonlinear branch and bound in only one step. The problem could also be slightly perturbed to $f(y) = (y - \delta)^2$ with $\delta < \frac{\epsilon}{2}$. Then starting at the solution $y^0 = 0$, linear outer approximation would again visit all feasible points before verifying that y^0 is the solution. Again

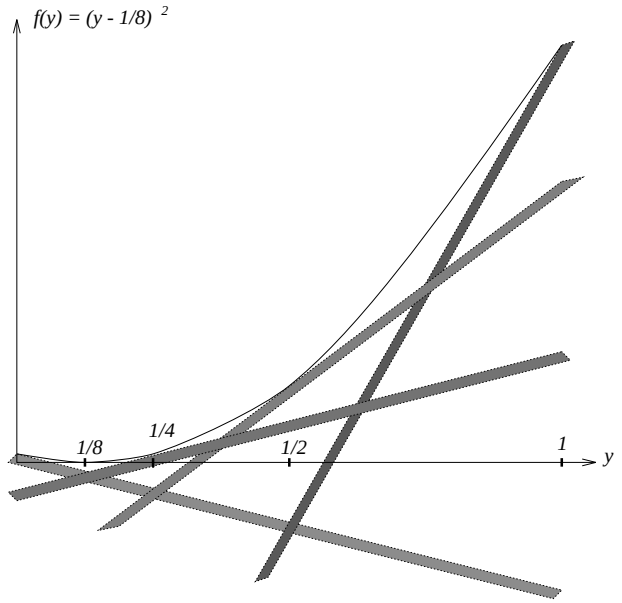

Figure 5.2: Worst case example for Algorithm 1

Generalized Benders Decomposition also visits all feasible points, but nonlinear branch and bound solves the problem after one branch.

The example shows that both linear outer approximation and Generalized Benders Decomposition perform badly when the problem functions are not adequately represented by linear approximations. The initial step makes the next iterate remote from the solution which is unsatisfactory in a nonlinear situation. The remedy lies in introducing curvature information into the master programs. In the remainder of this section it is shown how this can be achieved for linear outer approximation by including a second order Lagrangian term into the objective function of the MILP master programs.

These considerations have led to the development of a new algorithm based on the use of second order information. The development of such an algorithm seems contradictory at first sight, since quadratic functions do not provide underestimators of general convex functions. However, the present derivation allows the inclusion of a curvature term into the objective function of the MILP master problem. This quadratic term influences the choice of the next iterate by the algorithm without surrendering the finite convergence properties which rely on the fact that the feasible region of the master problem is an outer approximation of the feasible region of the MINLP problem P. The resulting algorithm is referred to as *quadratic outer approximation* and is obtained by replacing the relaxed master problem M^i by the problem

$$(Q^i)\begin{cases} \min\limits_{x,y,\eta} & \eta + \frac{1}{2}\begin{pmatrix} x-x^i \\ y-y^i \end{pmatrix}^T \nabla^2\mathcal{L}^i \begin{pmatrix} x-x^i \\ y-y^i \end{pmatrix} \\ \text{subject to} & \eta < \text{UBD}^i \\ & \eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x-x^j \\ y-y^j \end{pmatrix} & \forall j \in T^i \\ & 0 \geq g^j + [\nabla g^j]^T \begin{pmatrix} x-x^j \\ y-y^j \end{pmatrix} \\ & 0 \geq g^k + [\nabla g^k]^T \begin{pmatrix} x-x^k \\ y-y^k \end{pmatrix} & \forall k \in S^i \\ & x \in X,\ y \in Y \text{ integer} \end{cases}$$

in Step 4 of Algorithm 1. In the definition of Q^i the function

$$\mathcal{L}^i = \mathcal{L}(x^i, y^i, \lambda^i) = f(x^i, y^i) + (\lambda^i)^T g(x^i, y^i)$$

is the usual Lagrangian function. A detailed description of the algorithm is as follows.

Algorithm 2: Quadratic Outer Approximation

Initialization: y^0 is given; set $i = 0$, $T^{-1} = \emptyset$, $S^{-1} = \emptyset$ and $\text{UBD}^{-1} = \infty$.

REPEAT

1. Solve the subproblem NLP(y^i), or a feasibility problem F(y^i) if NLP(y^i) is infeasible, and let the solution be x^i.

2. Linearize the objective and (active) constraint functions about (x^i, y^i). Set $T^i = T^{i-1} \cup \{i\}$ or $S^i = S^{i-1} \cup \{i\}$ as appropriate and update the second order term $\nabla^2\mathcal{L}^i$.

3. **IF** (NLP(y^i) is feasible and $f(x^i, y^i) < \text{UBD}^{i-1}$) **THEN**
   update current best point by setting $x^* = x^i,\ y^* = y^i,\ \text{UBD}^i = f(x^i, y^i)$.
   ELSE
   Set $\text{UBD}^i = \text{UBD}^{i-1}$.

4. Solve the current relaxation Q^i of the master program M, giving a new integer assignment y^{i+1} to be tested in the algorithm. Set $i = i + 1$.

UNTIL (Q^i is infeasible).

Including a curvature term in the objective function does not change the finite convergence property expressed in Theorem 5.2.1, since the feasible region of M^i is unchanged and the constraints of Q^i are still supporting hyperplanes. However, the possibility of using the lower bound $\eta \geq \eta^i$ is no longer conveniently available. A quadratic Taylor series does not provide a lower bound on a convex function as the linear Taylor series does. Therefore it is not possible to use the optimal value of Q^i as a lower bound. Moreover it cannot even be expected that the optimal value of the linear part of the objective function (η) of the master problem relaxation provides a lower bound on P although η is only constrained by the supporting hyperplanes on f. This is illustrated by the

following example:

$$\begin{cases} \min\limits_{x} & f(x) = -\ln(x+1) \\ \text{s. t.} & 0 \leq x \leq 2. \end{cases}$$

For $x = 0$ the minimum of the quadratic approximation to f is at $x = \frac{1}{2}$ and the value of the linear approximation about $x = 0$ at $x = \frac{1}{2}$ is $-\frac{1}{2}$ which is greater than the minimum of the above problem $(-\ln(3))$.

The advantage of the quadratic outer approximation algorithm is that a possibly different selection y^{i+1} is made by solving the master problem Q^i, which takes into account nonlinear terms in P. This is well seen in the worst case example for linear outer approximation, which is solved by the quadratic outer approximation algorithm is two iterations, independent of p. The price that has to be paid for this better performance is that instead of solving an MILP at each iteration, an MIQP master program has to be solved. Unfortunately there is little or no software available that is specifically tailored to an MIQP problem. Ways to solve the resulting MIQP problems by a branch and bound strategy that uses improved lower bounds for child problems have been presented in Chapter 3 alongside a review of other methods to solve MIQPs. A Generalized Benders Decomposition approach for solving the MIQP as suggested by Lazimy [49] seems inadequate, since Generalized Benders Decomposition can again be interpreted as a linear model, in which case the difficulties caused by nonlinearities in outer approximation will simply arise at the MIQP level. Other methods that have been suggested include the branching rule of Körner [45] and a branching rule that was suggested by Breu and Burdet [9] for linear 0–1 programming.

The result of the numerical study in Chapter 6 suggests that it is not always advantageous to add a curvature term to the master program M^i. This has led to the derivation of an example where linear outer approximation solves the problem in one iteration, while quadratic outer approximation visits all integer assignments in turn. The example is

$$\begin{cases} \min\limits_{y} & f(y) = \exp(-y) \\ \text{subject to} & y \in \{0, 1, \ldots, n\} \end{cases}$$

where n is a positive integer. Starting at $y^0 = 0$, linear outer approximation finds the optimal solution in one iteration, while quadratic outer approximation visits all n integer points in turn before finding the optimum (since the minimum of a quadratic approximation to the objective about $y^i = i$ is $y^{i+1} = i+1$). In this example, quadratic outer approximation predicts nonexisting minima whereas linear outer approximation models the monotone objective function better. However, this situation is somewhat artificial in the sense that the starting point lies far from the final minimizer.

In order to gain further insight into the Algorithms 1 and 2 it is useful to consider the case when they are applied to pure integer nonlinear problems. Both algorithms make linear approximations to the constraints at y^i; the difference lies in the fact that the quadratic algorithm also includes a second order Lagrangian term in the next master program. Therefore quadratic outer approximation can be interpreted as a Sequential Quadratic Programming method generalized to integer programming. Linearizations of previous steps are kept in the master program to avoid cycling between successive integer assignments, and the QP subproblem of an ordinary SQP method is replaced by an MIQP problem to account for the discrete nature of the problem. The linear outer approximation algorithm can be interpreted as a Sequential Linear Programming technique.

This observation gives an indication as to when quadratic outer approximation should be preferred. If y appear only linearly in the problem, then it is hardly worthwhile to solve MIQP master programs. There is also unlikely to be much advantage in using quadratic outer approximation when the integer variables are mostly zero–one variables. The most favourable case for quadratic outer approximation occurs when there are multiple discrete values of each component of y, and

there are nonlinear terms in y present. However, care has to be taken, since nonlinearities can be hidden away by using linearization techniques that reformulate the original problem. Such a reformulation would indeed be necessary if one wanted to use Duran and Grossmann's outer approximation algorithm.

5.3 LP/NLP based branch and bound algorithm

This section presents a new approach to the solution of successive master program relaxations. It has been proposed by Quesada and Grossmann [60] for a class of problems whose objective and constraint functions are linear in the integer variables and is called *LP/NLP based branch–and–bound algorithm*. Their approach is generalized here to cover problems with nonlinearities in the integer variables. Moreover, the worst case example of Section 5.2 is also a worst case example for this algorithm. This motivates the investigation of a new *QP/NLP based branch and bound algorithm* which takes curvature information into account.

The motivation for the LP/NLP based branch and bound algorithm is that outer approximation usually spends an increasing amount of computing time in solving successive MILP master program relaxations. Since the MILP relaxations are strongly related to one another this means that a considerable amount of information is re-generated each time a relaxation is solved. The present approach avoids the re–solution of MILP master program relaxations by updating the branch and bound tree. This section makes extensive use of branch and bound terminology and the reader is referred to Chapter 3 for the relevant definitions.

Instead of solving successive relaxations of M, the new algorithm solves only one MILP problem which is updated as new integer assignments are encountered *during* the tree search. Initially an NLP–subproblem is solved and the initial master program relaxation M^0 is set up from the supporting hyperplanes at the solution of the NLP–subproblem. The MILP problem M^0 is then solved by a branch and bound process with the exception that each time a node (corresponding to an LP problem) gives an integer feasible solution y^i, say, the process is interrupted and the corresponding NLP(y^i) subproblem is solved. New linearizations from NLP(y^i) are then added to every node on the stack, effectively updating the branch and bound tree. The branch and bound process continues in this fashion until no problems remain on the stack. At that moment all nodes are fathomed and the tree search is exhausted.

Unlike ordinary branch and bound a node cannot be assumed to have been fathomed, if it produces an integer feasible solution, since the previous solution at this node is cut out by the linearizations added to the master program. Thus only infeasible nodes can be assumed to be fathomed. In the case of MILP master programs there exists an additional opportunity for pruning. Since the LP nodes are outer approximations of the MINLP subproblem corresponding to their respective subtree a node can be regarded as fathomed if its lower bound is less than or equal to the current upper bound UBD^i (defined as in Section 5.2).

The present algorithm also corrects an inaccuracy which occurs in [60] since it is based on the correctly reformulated MILP master program M, which includes a rigorous treatment of infeasible subproblems. Now the algorithm can be stated in full detail.

Algorithm 3: LP/NLP based branch and bound

Initialization: y^0 is given; set $i = 1$, $T^{-1} = \emptyset$, $S^{-1} = \emptyset$.

Set up the initial master program:

- Solve the subproblem NLP(y^0) and let the solution be x^0.
- Linearize the objective and (active) constraint functions about (x^0, y^0). Set $T^0 = \{0\}$.
- Set the upper bound and initialize the current best point. $x^* = x^0,\ y^* = y^0,\ \text{UBD}^0 = f^0$.

Place the master program M^0 with its integer restrictions relaxed on the stack and start the branch and bound process:

WHILE (stack is not empty) **DO BEGIN**

1. Remove the top problem (P') from the stack and solve the LP giving (x', y', η'). η' is a lower bound for all NLP son problems whose root is the current problem.

2. **IF** (y' integer) **THEN**

   (a) Set $y^i = y'$.

   (b) Solve the subproblem NLP(y^i), or a feasibility problem F(y^i) if NLP(y^i) is infeasible, and let the solution be x^i.

   (c) Linearize the objective and (active) constraint functions about (x^i, y^i). Set $T^i = T^{i-1} \cup \{i\}$ or $S^i = S^{i-1} \cup \{i\}$ as appropriate and add these linearizations to *all* pending problems on the stack.

   (d) **IF** (NLP(y^i) is feasible and $f(x^i, y^i) < \text{UBD}^i$) **THEN**
   update current best point by setting $x^* = x^i,\ y^* = y^i$,
   $\text{UBD}^{i+1} = f(x^i, y^i)$.
   Update the upper bound for all pending problems.
   ELSE
   Set $\text{UBD}^{i+1} = \text{UBD}^i$.

   (e) Place the problem (P') back on the stack and set $i = i + 1$.

   (f) *Pruning*: Remove all problems from the stack whose value η' is larger than UBD^{i+1}, since no better solution can be found in the corresponding subtree.

   ELSE

   - Branch on an integer variable and add two new problems to the stack.

   ENDIF

END (WHILE–LOOP)

As in the two outer approximation algorithms of Section 5.2 the use of an upper bound implies that no integer assignment is generated twice during the tree search. Since both the tree and the set of integer variables are finite the algorithm eventually encounters only infeasible problems and the stack is thus emptied so that the procedure stops. This provides a proof of the following corollary to Theorem 5.2.1.

Corollary 5.3.1 *If assumptions* **A1**, **A2**, **A3** *and* **A4** *hold, and* $|Y| < \infty$, *then Algorithm 3 terminates in a finite number of steps at a solution of P or with an indication that P is infeasible.*

The present algorithm implements a depth-first tree search. It is pointed out in [60] that the aim of this is to produce an integer assignment quickly, in order to tighten the master program early on. Backtracking is then performed to the most promissing node remaining on the stack. Figure 5.3 gives a schematic representation of the progress of the algorithm.

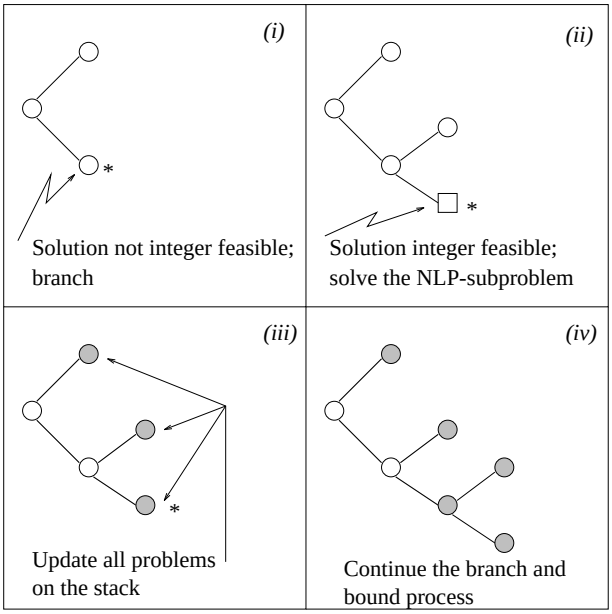

Figure 5.3: Progress of the LP/NLP based branch and bound algorithm

In Figure 5.3 *(i)*, the LP relaxation of the initial MILP has been solved and two branches added to the tree. The LP that is solved next (indicated by an *) does not give an integer feasible solution and two new branches are introduced. The next LP in Figure 5.3 *(ii)* produces an integer feasible solution indicated by a □. The corresponding NLP subproblem is solved and in Figure 5.3 *(iii)* all nodes on the stack are updated (indicated by the shaded circles) by adding the linearizations from the NLP subproblem including the upper bound UBD^i which cuts out the current assignment y^i. Then, the branch and bound process continues on the updated tree by solving the LP marked by a *. If this results in a ∘ node then the algorithm continues by branching. If a □ node again results then a new NLP subproblem is solved.

It is instructive to consider applying the LP/NLP based branch and bound algorithm to the worst case example of Section 5.2. Starting at $y^0 = 0$, the next iterate is $y^1 = 1$ and as in linear outer approximation, Algorithm 3 then works its way back to the solution visiting each remaining integer assignment in turn. This example also illustrates the need to re–solve the LP nodes that are integer feasible. The initial LP relaxation is integer feasible, giving $y^1 = 1$. If it would not be re–solved after the new supporting hyperplane has been added, the algorithm would terminate with the incorrect solution $y^* = 0$.

The example reveals the inability of Algorithm 3 to take curvature information into account. As in Section 5.2 this can be remedied by including a curvature term into the objective function of the relaxed MILP master programs. The resulting algorithm is termed *QP/NLP based branch and bound algorithm*. It differs from Algorithm 3 in two important aspects. The first difference is that

QP rather than LP problems are solved in the tree search. The second difference is a consequence of the first. Since QP problems do not provide lower bounds on the MINLP problems P, the pruning step in Algorithm 3 cannot be applied. A detailed description of the new algorithm is as follows.

Algorithm 4: QP/NLP based branch and bound

Initialization: y^0 is given; set $i = 1$, $T^{-1} = \emptyset$, $S^{-1} = \emptyset$.

Set up the initial master program Q^0:

- Solve the subproblem NLP(y^0) and let the solution be x^0.
- Linearize the objective and (active) constraint functions about (x^0, y^0). Set $T^0 = \{0\}$ and compute $\nabla^2\mathcal{L}^0$.
- Set the upper bound and initialize the current best point. $x^* = x^0,\ y^* = y^0,\ \text{UBD}^0 = f^0$.

Place the master program Q^0 *with its integer restrictions relaxed on the stack and start the branch and bound process:*

WHILE (stack is not empty) **DO BEGIN**

1. Remove the top problem (P') from the stack and solve the QP giving (x', y', η').
2. **IF** (y' integer) **THEN**
   (a) Set $y^i = y'$.
   (b) Solve the subproblem NLP(y^i), or a feasibility problem F(y^i) if NLP(y^i) is infeasible, and let the solution be x^i.
   (c) Linearize the objective and (active) constraint functions about (x^i, y^i). Set $T^i = T^{i-1} \cup \{i\}$ or $S^i = S^{i-1} \cup \{i\}$ as appropriate and add these linearizations to *all* pending problems on the stack. Update the second order term by computing $\nabla^2\mathcal{L}^i$ for *all* pending problems on the stack.
   (d) **IF** (NLP(y^i) is feasible and $f(x^i, y^i) < \text{UBD}^i$) **THEN**
   update current best point by setting $x^* = x^i,\ y^* = y^i$,
   $\text{UBD}^{i+1} = f(x^i, y^i)$.
   Update the upper bound for all pending problems.
   ELSE
   Set $\text{UBD}^{i+1} = \text{UBD}^i$.
   (e) Place the problem (P') back on the stack, set $i = i + 1$.

   ELSE

   - Branch on an integer variable and add two new problems to the stack.

   ENDIF

END (WHILE–LOOP)

It should be noted that any branching rule mentioned in Chapter 3 can be used in this branch and bound process, though the rule based on improved lower bounds for MIQP is not recommended, since the MIQP master program does not provide a lower bound on the MINLP problem. The branching rule that is used here is the maximal fractional part branching rule introduced in Chapter 3.

5.4 Implementation Issues

This section addresses some implementation issues arising from the Algorithms 1 to 4. In particular it is shown how a more flexible formulation of P can be treated and how the algorithms can be started, if no initial integer assignment is available. Moreover it is indicated that different Hessian matrices could be used in the MIQP master program relaxations.

In practice one would not expect to encounter P in the form given, but rather in the more general form

$$\text{P'} \begin{cases} \min\limits_{x} & f(x) \\ \text{s. t.} & l_g \leq g(x) \leq u_g \\ & l_x \leq x \leq u_x \\ & x_i,\ i \in I \ \text{ integer} \end{cases}$$

The treatment of integer variables occurring anywhere in the variable vector is standard and the distinction of x and y has only been made to improve the readability of the previous two chapters. There is no additional difficulty involved in dealing with problems of this type. Upon termination, NLP solvers provide the user with an indication as to which particular bounds are active and it suffices to add the linearizations of these active bounds to the master program. However, no guarantee can be given that the algorithms will find the solution if P' is not a convex programming problem although all algorithms will terminate finitely.

All four algorithms require an initial integer assignment to be given. If this is not easily available, then Viswanathan and Grossmann [73] suggest to start the algorithm by solving the NLP relaxation of P, which is obtained by relaxing all integer restrictions. This corresponds to solving the root problem in an NLP branch and bound algorithm. If the relaxed NLP is integer feasible, then this solution also solves P and the algorithm stops. Likewise, if the relaxed NLP is infeasible, then so is P and the algorithm stops with an indication that P is infeasible. If the relaxed NLP has a feasible solution which is not integer, then the relaxed NLP is linearized about its solution and these linearizations form the initial MILP or MIQP master program relaxation.

It is important to realize that Algorithms 3 and 4 can be implemented very efficiently in practice. Since all LP or QP problems on the stack share the same objective and the same feasible region it is not necessary to store these individually with each problem. Instead, as in nonlinear branch–and–bound it suffices to store the lower and upper bounds of the integer variables and possibly an advanced basis for each problem on the stack. Thus, an implementation of the LP/NLP or QP/NLP based branch–and–bound algorithm is as efficient as an implementation of nonlinear branch–and–bound in terms of storage requirements.

Finally, it is possible to modify the MIQP based Algorithms 2 and 4 slightly by varying the way in which the curvature information is changed. An obvious way is to update the curvature term whenever a new feasible NLP has been solved. However, it might be advantageous to update only when an improved integer solution has been found. The reasoning behind this approach being that it is usually important to examine the neighbourhood of the solution before enough supporting hyperplanes have been generated and the algorithm can finish.

5.5 An Outer Approximation Algorithm based on Lagrangean Decomposition

Lagrangean Decomposition was introduced by Michelon and Maculan [54] for nonlinear integer programming problems with linear constraints, extending results given by Guignard and Kim [34] for MILP Lagrangean Decomposition. Their theoretical results are presented in a similar way to Geoffrion's [27] analysis of Lagrangean methods. This section generalizes the approach taken by

Michelon and Maculan to include problems of type P with nonlinear constraints. First a Lagrangean Decomposition is derived and a dual problem to P is introduced. The dual problem decomposes into two independent problems, an MILP and an NLP. Weak duality relations are derived and an algorithm is presented that aims at reducing the duality gap. In order to reduce the duality gap, the proposed algorithm uses supporting hyperplanes in a similar fashion to the algorithms of the previous section. A finite convergence result is given and it is indicated how the algorithm is likely to perform compared to the outer approximation algorithms of the previous sections.

In order to improve the readability of the material exposed in this section, it is convenient, to combine the continuous variables x and the integer variables y into one vector, denoted y again. The vector x is thus reserved for the copied set of variables for which no integer restrictions apply. The model problem then becomes

$$\mathrm{P}_{LD} \begin{cases} \min\limits_{y} & f(y) \\ \text{subject to} & g(y) \leq 0 \\ & A^T y \leq b \\ & y \in Y, \end{cases}$$

where f and g are convex continuously differentiable functions on $conv(Y)$, the convex hull of the set $Y \subset \mathbb{R}^n$ which contains the integer restrictions. Note that the linear constraints appear separately in order to enable a simple derivation of $conv(Y)$ to be used.

First a new variable x which has the role of a continuous copy of y is introduced by adding copy constraints $x = y$ to P_{LD}. The continuous copy x replaces y in the nonlinear functions f and g and copies of the linear constraints ($A^T x \leq b$ and $x \in conv(Y)$) are also added to P_{LD}, and this gives rise to the following equivalent problem.

$$\begin{cases} \min\limits_{x,y} & f(x) \\ \text{subject to} & g(x) \leq 0 \\ & A^T x \leq b \\ & x \in conv(Y) \\ & x = y \\ & A^T y \leq b \\ & y \in Y. \end{cases}$$

This problem is now dualized with respect to the copy constraints $x = y$. Let $u \in \mathbb{R}^n$ denote the corresponding multiplier, then a Lagrangian problem can be defined as

$$l(u) = \begin{cases} \min\limits_{x,y} & f(x) - u^T(x - y) \\ \text{subject to} & g(x) \leq 0 \\ & A^T x \leq b \\ & x \in conv(Y) \\ & A^T y \leq b \\ & y \in Y. \end{cases}$$

The reason for this dualization now becomes apparent in that for fixed u the problem $l(u)$ decouples into two separate problems, that is

$$l(u) = \begin{cases} \min\limits_{x} & f(x) - u^T x \\ \text{subject to} & g(x) \leq 0 \\ & A^T x \leq b \\ & x \in conv(Y) \end{cases} + \begin{cases} \min\limits_{y} & u^T y \\ \text{subject to} & A^T y \leq b \\ & y \in Y. \end{cases}$$

This decomposition will be referred to as *Lagrangian Decomposition* in the remainder of this section. Given $u \in \mathbb{R}^n$, $l(u)$ is computed by solving the following two independent problems.

$$\mathrm{P}_{ux} \left\{ \begin{array}{ll} \min\limits_{x} & f(x) - u^T x \\ \text{subject to} & g(x) \leq 0 \\ & A^T x \leq b \\ & x \in conv(Y), \end{array} \right.$$

and

$$\mathrm{P}_{uy} \left\{ \begin{array}{ll} \min\limits_{y} & u^T y \\ \text{subject to} & A^T y \leq b \\ & y \in Y. \end{array} \right.$$

Problem P_{ux} is an ordinary NLP for which the integer restrictions are relaxed and P_{uy} is an MILP. Both problems are coupled only through the multiplier u. Thus Lagrangian Decomposition employs the same basic idea used by Outer Approximation or Benders' Decomposition of separating the nonlinear and the combinatorial part of the MINLP problem.

An important property of Lagrangian Decomposition is that $l(u)$ is the value of a dual problem of P_{LD} for every $u \in \mathbb{R}^n$, thereby providing a lower bound on the optimum value of P_{LD}. In particular, if $v(P)$ denotes the optimal value of a problem P and y^* is the optimal solution to P_{LD} then

$$l(u) \leq f(y^*) = v(\mathrm{P}_{LD}) \ , \ \forall u \in \mathbb{R}^n,$$

since $x = y^*$ is feasible in $l(u)$, $\forall u$. This observation motivates the introduction of the following weak dual to P_{LD}

$$\mathrm{D}_{LD} \left\{ \max_{u \in \mathbb{R}^n} l(u). \right.$$

Usually there will be a duality gap (i.e. $v(\mathrm{D}_{LD}) < v(\mathrm{P}_{LD})$) and solving the dual is not always satisfactory. The aim of the algorithm which is proposed towards the end of this section is to reduce this duality gap by adding supporting hyperplanes to the MILP problem P_{uy} similar to the algorithm proposed in the previous section.

It is of interest to compare the dual D_{LD} with the problem P'_{LD} obtained from P_{LD} by relaxing all integer restrictions. The following manipulations imply that the dual D_{LD} is stronger than P'_{LD} in the sense that the former gives at least as good a lower bound as the latter. Taking $u = 0$ it follows that

$$l(0) = v(\mathrm{P}_{0x}) = v(\mathrm{P}'_{LD})$$

which implies that

$$v(\mathrm{D}_{LD}) = \max_{u \in \mathbb{R}^n} l(u) \geq l(0) = v(\mathrm{P}'_{LD}).$$

Lagrangean Decomposition is especially interesting if special purpose routines are available to solve the two independent problems P_{ux} and P_{uy}. Michelon and Maculan propose two algorithms that aim at reducing the duality gap iteratively. Their first algorithm is of theoretical interest only, since it includes an additional nonlinear constraint in P_{uy} to cut off points y that have been visited before or that are not promising. The nonlinear constraint that is included in their MILP problem is

$$f(y) \leq \mathrm{UBD} - \epsilon$$

where UBD is an upper bound. Their second algorithm replaces this nonlinear constraint by a collection of supporting hyperplanes, obtaining a constraint set similar to the master program

relaxation of the outer approximation algorithms of Section 5.2. This last observation motivated the generalization of Lagrangean Decomposition to MINLP problems with nonlinear constraints.

Now the generalization of Michelon and Maculan's second algorithm is described.

Algorithm 5: Lagrangean Decomposition

Initialization: Set $i = 0$, $u^0 = 0$, UBD$= \infty$ and LBD$= -\infty$.

REPEAT

1. Solve the NLP problem P_{ux} for $u = u^i$ and obtain x^i.

2. Solve the MILP problem

$$M_{u^i y} \begin{cases} \min\limits_{y} & (u^i)^T y \\ \text{subject to} & f^j + (\nabla f^j)^T (y - y^j) \leq \text{UBD} - \epsilon \\ & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j = 0, 1, \ldots, i \\ & g^j + [\nabla g^j]^T (y - y^j) \leq 0 \\ & A^T y \leq b \\ & y \in Y, \end{cases}$$

   and obtain y^i.

3. Update the lower and upper bounds
   IF ($g(y^i) \leq 0$ and $f(y^i) <$UBD) **THEN** UBD$= f(y^i)$, $y^* = y^i$
   IF ($l(u^i) >$LBD) **THEN** LBD$= l(u^i)$

4. Update the multipliers by setting $u^{i+1} = u^i + t_i(y^i - x^i)$. Set $i = i + 1$.

UNTIL (UBD – LBD $< \epsilon$)

Before stating and proving the convergence result it is instructive to discuss the algorithm in more detail. If the initial NLP problem P_{ux} for $u^0 = 0$ is infeasible, then the algorithm can stop, since P_{LD} would also be infeasible.

The multiplier u^i can be updated by any current technique available. Nemhauser and Wolsey [56] give two schemes for the parameter $\{t_i\}_{i=0,1,2,\ldots}$. The parameters can be from a divergent series such that

$$\sum_{i=0}^{\infty} t_i \to \infty, \; t_i \to 0 \;\text{ as } i \to \infty$$

or from a geometric series such that

$$t_i = t_0 r^i \;\text{ or } t_i = \frac{(f' - f(y^i))r^i}{\|x^i - y^i\|^2},$$

where $0 < r < 1$ and f' is an upper bound on the optimal solution of P_{LD}. Geoffrion [25] shows that the optimal multipliers of an NLP problem can be identified with the subgradient of the value function (Theorem 1, [25]). Thus, it is possible to interpret Lagrangean Decomposition as a subgradient method for solving the dual. Alternatively, it is possible to solve D_{LD} by bundle methods (e.g. [66]) which are likely to be a better approach than subgradient methods.

Michelon and Maculan include an additional stopping criterion

$$\|y^i - x^i\| < \epsilon_1 \tag{5.5}$$

in their algorithm. It is not difficult to deduce that this is redundant, since (5.5) implies the stopping criterion used in Algorithm 5. Assume that (5.5) holds. It follows then that

$$\text{UBD} \leq f(y^i) \;\text{ and }\; \text{LBD} \geq l(u^i),$$

so that

$$\begin{aligned} \text{UBD} - \text{LBD} &\leq f(y^i) - l(u^i) \\ &= (u^i)^T(x^i - y^i) + f(y^i) - f(x^i) \\ &\leq \|u^i\| \cdot \|x^i - y^i\| + |f(y^i) - f(x^i)| \\ &\leq \|u^i\| \cdot \epsilon_1 + \epsilon_2 \\ &\leq \epsilon, \end{aligned}$$

provided that the sequence of multipliers $\{u^i\}_{i=0,1,2,\ldots}$ is bounded.

The "master problem" that is solved in Lagrangian Decomposition is very similar to the one solved in outer approximation. The difference lies in the objective function which contains the multiplier and in the fact that the constraints are added to the master program *without* regard for the feasibility of y^j. The multipliers also work as the connection between the "master program" and the NLP problem, where unlike in outer approximation no copy constraints are present to fix the integer variables. Consequently, the NLP problem that is solved at each iteration of Algorithm 5 is larger than the NLP subproblem in outer approximation. Apart from this, both algorithms are equally expensive per iteration, solving both one MILP and one NLP problem per iteration.

The lower bounds in the Lagrangean Decomposition algorithm are derived from the duality relation and *not* from the optimal value of the master program relaxations as in outer approximation. The fact that the upper bounds are obtained by simply evaluating the objective function at y^i means that they are not as sharp as in outer approximation in the mixed integer case where in addition a minimization problem in the continuous variables is solved (the NLP–subproblem).

The most important difference, however, between outer approximation and Lagrangean Decomposition is the fact that although it can be shown that no point $y \in Y$ is generated twice, it is possible in the case of mixed–integer problems that the same *integer* assignment is generated twice by Lagrangean Decomposition albeit with *different* continuous parts. The reason for this undesirable feature is that Lagrangean Decomposition obtains the upper bound by evaluating f at the solution of the master program, whereas outer approximation solves a minimization problem in the x variables. As a consequence, I do not regard Lagrangean Decomposition as a valuable alternative to outer approximation.

After this brief discussion of the proposed algorithm, a finite convergence proof can now be given. The proof is similar to the one set out in [54] for linearly constrained problems. Improving on the proof in [54] the present proof also proves termination at an ϵ–optimal solution of P_{LD}. A similar result holds if other constraint qualifications than **A3** are used.

Theorem 5.5.1 *Let f and g be convex continuously differentiable functions on conv(Y) and assume that g satisfies the constraint qualification* **A3**. *Then Algorithm 5 converges in a finite number of steps to an optimal solution of P_{LS}.*

Proof:
First it is shown that the algorithm terminates finitely. There are two cases that are considered separately depending on whether or not Y is a finite set or not.

(i) If $Y \subset \mathbf{Z}^n$ so that $|Y| < \infty$ then the finiteness follows from the finiteness results established for outer approximation (c.f. Theorem 5.2.1).

(ii) Now consider the case where $Y \subset \mathbb{R}^n$ is a set with integer restrictions, but not finite. It is assumed that Algorithm 5 does *not* terminate finitely and a contradiction is sought. If Algorithm 5

is not finite then there exists a sequence of iterates $\{y^i\}_{i=0,1,2,\ldots}$ that is generated by the algorithm. Since Y is a bounded and closed set it follows that there exists a subsequence, say $\{z^l\}_{l=0,1,2,\ldots}$ which converges to a limiting point $z' \in Y$ (Theorem of Bolzano and Weierstraß, e.g. Forster [23]). At this point it becomes necessary to distinguish again between two cases since the method of proof depends on whether the limit z' is feasible or not.

(ii-a) If the limit of the subsequence, z', is not feasible then there exists an index p such that $g_p(z') > 0$. The continuity of g implies that there exists an index l_0 such that

$$g_p(z^l) > 0, \ \forall l \geq l_0$$

Now let z^i and z^j be two points from the sequence such that $l_0 < i < j$. The feasibility of z^j in the "master program" at iteration i implies that

$$0 \geq g_p^i + (\nabla g_p^i)^T (z^j - z^i)$$

Rearranging and applying Cauchy Schwartz inequality (e.g. Stammbach [68]) gives

$$g_p^i \leq (\nabla g_p^i)^T (z^i - z^j) \leq \|\nabla g_p^i\| \cdot \|(z^j - z^i)\|$$

Since g_p is continuously differentiable it follows that

$$\lim_{i \to \infty} \|\nabla g_p^i\| = \|\nabla g_p'\|.$$

Furthermore

$$\lim_{i \to \infty, \, j \to \infty} \|(z^j - z^i)\| = 0,$$

so that

$$g_p' \leq 0$$

which contradicts the infeasibility of z'.

(ii-b) The second case occurs if the limit z' is a feasible point. Again let z^i and z^j be two points from the sequence such that $i < j$. The feasibility of z^j in the "master program" at iteration i implies now that

$$f^i + (\nabla f^i)^T (z^j - z^i) \leq \text{UBD} - \epsilon,$$

and $\text{UBD} \leq f^i$ so that the following inequality is obtained after rearranging, substituting for UBD and cancelling f^i. Applying the Cauchy Schwartz inequality then gives

$$\epsilon \leq (\nabla f^i)^T (z^i - z^j) \leq \|\nabla f^i\| \cdot \|z^j - z^i\|.$$

Since f is continuously differentiable it follows that

$$\lim_{i \to \infty} \|\nabla f^i\| = \|\nabla f'\|.$$

And similarly to (ii-a) it follows that

$$\epsilon \leq 0,$$

which is a contradiction and concludes the finite termination proof for the mixed integer case.

It remains to prove that Algorithm 5 terminates at an ϵ–optimal solution of P_{LD}. Assume that Algorithm 5 terminates at iteration k with a solution y' which is not ϵ–optimal, so that $f' > f^* + \epsilon$ where $f^* = v(\text{P}_{LD})$. Then it follows that

$$\text{UBD} = f' > f^* + \epsilon > \text{LBD} \geq \max_{j=0,1,\ldots,k} l(u^j)$$

and there exists still a duality gap. The convexity of f and g implies that y^* is feasible in the master program of iteration $k+1$ so no stop can have occurred, which contradicts the assumptions and concludes the finite convergence proof.

□

Chapter 6

Numerical Testing of the MINLP Routines

6.1 Introduction

The Algorithms presented in Chapter 5 share many of the same theoretical properties. In order to make any comments on their respective practical merits it is, therefore, necessary to test the practical performance of the routines. This chapter contains the description of the numerical study and a discussion of its results. The first section gives a description of the implementation, the performance meters and the test problems. The outcome of the tests are listed and discussed in the next section.

Five different MINLP solvers are tested in this chapter. The first is an implementation of a nonlinear branch–and–bound routine as described in Chapter 3. The remaining solvers implement the linear and quadratic outer approximation algorithm and the LP/NLP and QP/NLP based branch–and-bound algorithm. All code is written in FORTRAN 77 and uses the NAG library routine E04UCF [55] to solve the NLP problems.

The implementation of the nonlinear branch–and–bound algorithm uses a depth–first–search with backtracking to the most promising node. The code branches on the most fractional variable first, thereby implementing the branching rule that is favoured in [36]. The nonlinear solver allows only one degree of "warm start" and this is used throughout the tree–search apart from the initial node. The code is written in double precision FORTRAN 77, since the available version of the NAG library is double precision.

The implementation of the outer approximation routines uses the same NLP solver as the nonlinear branch–and–bound routine. An MILP/MIQP branch–and–bound code has also been written and this is used to solve the master program relaxations. The MILP/MIQP branch–and–bound code implements a depth–first–search with backtracking to the most promising node and branches on the most fractional variable first. The individual LP/QP problem is solved using bqpd [18] which resolves degeneracy and has a guaranteed termination even in the presence of round–off errors. The solver is efficient for both MILP and MIQP problems, since no quadratic information is computed in the MILP case. The QP/LP solver bqpd has several modes of "cold", "warm" and "hot start" which are used whenever possible during the tree search. If the problem is binary, then integer cuts are added to exclude any integer assignment from being generated again.

The implementation of the LP/NLP and QP/NLP based branch–and–bound algorithms of Chapter 5 uses the branching rule that branches on the most fractional variable first. Similar to the nonlinear branch–and–bound code a depth first search is used with backtracking to the most promising node. The QP/LP problems are solved by bqpd. If no curvature information is added

to the master program, then lower bounds derived from the value of the previous master program are included in the current master program.

All routines require the user to specify the objective and the nonlinear constraint functions together with their gradients in a separate subroutine. The user must also provide a data file containing the dimensions, the linear constraints, the simple bounds, the bounds on the constraints and the indices of the integer variables. If curvature information is included, then a subroutine that computes the Hessian of the Lagrangian (or an approximation) is also required.

The performance meters that are used to compare the algorithms are given in the following list.

- Total CPU time. This measure is only exact to within about 5% and excludes compilation time, data input time and data output time.
- Total number of QP/LP problems solved. This counts the number of QP/LP problems solved by the MIQP/MILP branch–and–bound routine plus the number of QP problems solved by the NLP solver.
- The number of NLP solves. This is not as objective as the previous two measures since a nonlinear branch–and–bound routine is expected to solve more NLPs while at the same time making better use of hot start facilities.

All three routines are started by solving an initial NLP relaxation of the MINLP problem, so that all three routines have the same starting point. All routines are terminated upon finding an ϵ–optimal solution. This implies that there is no unfair disadvantage for the nonlinear branch–and–bound routine which cannot take advantage quite as readily of a starting point (it can take advantage of a starting point as a means of providing an initial upper bound).

Problem	n_i	n_c	m_e	m_i	m_n	integers	description
TP1	3	3	0	4	2	binary	synthesis problem, [15]
TP2	6	5	0	11	3	binary	synthesis problem, [15]
TP3	9	8	0	19	4	binary	synthesis problem, [15]
BATCH	24	22	12	60	1	binary	multiproduct batch plant, [42]
BATCH–int	6	16	0	30	31	integer	multiproduct batch plant, [42]
Asaadi 1 (3)	3	1	0	0	3	integer	$x_i, i = 1, 2, 4$ integer, [2]
Asaadi 1 (4)	4	0	0	0	3	integer	pure integer, [2]
Asaadi 2 (4)	4	3	0	0	4	integer	$x_i, i = 1, \ldots, 4$ integer, [2]
Asaadi 2 (7)	7	0	0	0	4	integer	pure integer, [2]
Asaadi 3 (6)	6	4	0	3	5	integer	$x_i, i = 1, 3, 5, 7, 8, 9$ integer, [2]
Asaadi 3 (10)	10	0	0	3	5	integer	pure integer, [2]
2DEx	2	0	0	0	2	discrete	2-D example, [10]
GTD	4	0	0	0	0	integer	Gear Train Design, [65]
GTD chain(2)	5	2	0	0	0	integer	chained Gear Train Design
GTD chain(3)	8	2	0	0	0	integer	chained Gear Train Design
AVGAS 1	8	0	0	10	0	binary	Hessian $G = tri(-1, 4, -1)$
AVGAS 2	8	0	0	10	0	binary	Hessian $G = tri(1, 4, 1)$
AFIRO (3)	3	29	8	19	0	integer	Hessian pos. def.
AFIRO (5)	5	27	8	19	0	integer	Hessian pos. def.
AFIRO (6)	6	26	8	19	0	integer	Hessian pos. def.
AFIRO (7)	7	25	8	19	0	integer	Hessian pos. def.

Table 6.1: Description of MINLP Test Problems

Table 6.1 describes the main characteristics of the test problems. The first column gives the name of the problem. Columns 2 to 6 give n_i, the number of integer variables, n_c the number of continuous variables, m_e the number of linear equality constraints, m_i the number of linear equality constraints and m_n the number of nonlinear constraints. The specification of the integer variables is given in column 7 followed by a description of the problem and its source in the last column.

Problems TP1, TP2 and TP3 are three small process synthesis problems in which the binary variables occur only linearly. BATCH is a problem derived from the optimal design of a multiproduct batch plant with 6 stages and 5 products. All these problems are convex MINLP problems. BATCH–int is the same problem as BATCH, but the integer variables modelling the number of parallel units per stage are not replaced by binary variables, hence the smaller problem size. BATCH–int is *not* convex (c.f. Appendix A), but was added to illustrate the effect of replacing integer variables by binary variables.

Problems Asaadi 1 through to Asaadi 3 are convex NLP test problems with quadratic objective and constraint functions. 2DEx is a small nonconvex MINLP problem with discrete variables. GTD is derived from designing a gear train with four gears to match a given gear ratio as good as possible in the l_2 sense, it is a nonconvex MINLP with box constraints. This problem can also be chained to give a larger mixed integer problem. The integer in "chain(2)" indicates the length of the chain. Finally, AVGAS and AFIRO are well known LP problems from the SOL test set. By adding a convex quadratic term into their objective they become convex MIQP problems.

6.2 Results and Discussion

The results of the tests are presented in Tables 6.2, 6.3 and 6.5. The test problems fall into two broad classes and the main point is that the five methods perform differently on each class. Consequently there is no one solver which can be seen to outperform all other solvers and it appears that a careful choice of solver gives the best result. A rule of thumb on which solver to choose for which problem is given at the end of this section.

The first table gives the number of NLP problems needed to solve the test problems with the five different solvers, the second table lists the number of QPs (including LPs) solved and the last table lists the total amount of CPU time used in the solves. The first column of each table gives again the name of the problem, the subsequent five columns give the performance data for the five routines tested. The heading nonlin–BB refers to nonlinear branch–and–bound, lin–OA refers to linear outer approximation, qua–OA refers to quadratic outer approximation, LP/NLP refers to the LP/NLP based branch–and–bound routine and QP/NLP refers to the QP/NLP based branch–and–bound algorithm.

There are no results for the quadratic outer–approximation code for the problems BATCH, GTD and GTD chain(3). This is due to severe growth of round–off error in the reduced Hessian matrix that can affect the QP solver `bqpd`. The solve for BATCH was aborted after more than an hour without obtaining a solution and the solves for the GTD problems were so inefficient that it was felt that they would otherwise distort the results. It is anticipated that this problem will be remedied in a future release of `bqpd`.

Table 6.2 shows that – as expected – nonlinear branch–and–bound usually requires more NLP solves than the outer approximation routines. Branch–and–bound searches the tree by solving NLP problems whereas the outer approximation routines search the tree by solving a quadratic or a linear model and rely only on a few NLP solves for given integer assignments. Yet nonlinear branch–and–bound can be more efficient than any outer approximation routine both in terms of the total number of QPs solved and in terms of the amount of CPU time required for a solve. The better CPU time is partly due to the fact that nonlinear branch–and–bound can make use of

Problem	nonlin–BB	lin–OA	qua–OA	LP/NLP	QP/NLP
TP1	5	4	4	4	4
TP2	13	4	4	5	7
TP3	20	7	5	8	9
BATCH	29	3		3	3
BATCH–int	11	2	2	2	2
Asaadi 1 (3)	3	2	2	2	2
Asaadi 1 (4)	7	3	3	5	3
Asaadi 2 (4)	7	19	6	6	5
Asaadi 2 (7)	30	12	13	11	20
Asaadi 3 (6)	27	15	9	21	11
Asaadi 3 (10)	138	12	9	22	11
2DEx	3	2	2	2	2
GTD	57	14	21	12	9
GTD chain(2)	11	2	2	2	2
GTD chain(3)	33	7		8	3
AVGAS 1	9	6	7	5	6
AVGAS 2	19	7	7	6	8
AFIRO (3)	8	6	6	7	4
AFIRO (5)	38	8	4	8	7
AFIRO (6)	73	11	8	10	8
AFIRO (7)	48	8	6	9	7

Table 6.2: Number of NLPs solved

"warm start" facilities and only one NLP is solved from scratch, whereas *all* NLPs have to be solved from scratch in the case of the outer–approximation routines. However, by considering the results for AFIRO (7) it can be seen that nonlinear branch–and–bound solves twice as many QPs as the QP/NLP based branch–and–bound routine in the same time. This is only partly due to the "warm starts" of the NLP solver and highlights the fact that two different QP solvers are used. While `bqpd` resolves degeneracy, the QP solver of the NAG library that underlies the NLP solver has no facility to detect or handle degeneracy and is consequently cheaper than `bqpd`. On one very large example, the NLP solver fails to find a feasible point of the *linear* constraints, due to degeneracy, while `bqpd` has no difficulty to locate a feasible point. The different degrees of reliability of the solvers should be kept in mind when comparing the results of the experiment and it is preferred to compare the routines by looking at the number of LPs and QPs solved.

If many NLP-subproblems are infeasible, the outer approximation routines suffer from a fault in the NLP solver which puts these routines at a disadvantage compared to nonlinear branch–and-bound. The reason for this fault is that the NLP–solver does not provide a solution to a phase I problem, like those presented in Section 4.2.2. Hence, it is necessary to solve an additional NLP feasibility problem. Ideally, one would prefer to avoid this, but this requires an NLP–solver which solves a phase I problem before entering into the optimization mode.

The two outer approximation routines which solve MILP or MIQP master programs, linear OA and quadratic OA, might become more efficient, if commercial MI–solver were used to solve the master program relaxations. In particular, for the binary problems, TP1 to TP3 and BATCH, where a commercial package would be expected to be more efficient, this could be a drawback. However, since the number of NLP solves is only slightly better than for the LP/NLP or QP/NLP based branch–and–bound algorithm, it is questionable whether a commercial MI–package would

Problem	nonlin–BB	lin–OA	qua–OA	LP/NLP	QP/NLP
TP1	28	37	33	27	28
TP2	71	84	80	47	59
TP3	122	167	148	82	89
BATCH	273	132		55	75
BATCH–int	154	69	74	51	63
Asaadi 1 (3)	16	17	16	15	15
Asaadi 1 (4)	52	71	51	36	31
Asaadi 2 (4)	69	616	197	137	104
Asaadi 2 (7)	176	2595	1168	407	253
Asaadi 3 (6)	182	1289	412	258	208
Asaadi 3 (10)	761	6217	2256	839	806
2DEx	9	31	19	19	17
GTD	165	358		345	94
GTD chain(2)	62	24	29	22	26
GTD chain(3)	259	162		103	51
AVGAS 1	30	48	101	25	32
AVGAS 2	40	71	103	21	33
AFIRO (3)	59	93	78	69	35
AFIRO (5)	671	777	247	244	87
AFIRO (6)	727	2480	750	331	213
AFIRO (7)	406	1517	478	300	172

Table 6.3: Number of LPs and QPs solved

make enough difference to outperform the latter two routines.

Comparing the results for TP1 to TP3 with those obtained by Duran and Grossmann [15] for the same problems, shows how much nonlinear branch–and–bound is improved by implementing the branching rule that branches on the most fractional variable first. Table 6.4 shows the number of NLPs solved for the three test problems for the branch–and–bound implementation of Duran and Grossmann and the present nonlinear branch–and–bound solver. The nonlinear branch–and–bound routine which branches on the most fractional variable first is almost twice as good as the corresponding solver used by Duran and Grossmann. This observation implies that the difference between a good implementation of the branch–and–bound algorithm and the outer approximation routines is not as big as originally thought.

	Number of NLPs solved	
Problem	Duran and Grossmann	This thesis
TP1	8	5
TP2	25	13
TP3	43	20

Table 6.4: Number of NLP solves for nonlinear branch–and–bound

Problem BATCH and BATCH–int have been solved in order to get an indication as to how the replacement of integer variables by binary variables affects the efficiency of the MINLP solvers. Although BATCH–int is *not* a convex problem all five solvers locate the optimal solution in about 20% of the time required for the equivalent BATCH. Thus in this example the replacement of integer variables results in a loss of efficiency.

The test problems fall into two large classes of problems on which the performance of the MINLP routine differs. The first class of problems is made up of the binary problems, TP1 to TP3, BATCH and the AVGAS problems. Apart from AVGAS, all other problems in this class have an objective function which is linear in the integer variables y and the nonlinear functions are all monotone. All other test problems fall into the second class of general integer MINLP problems.

Problem	nonlin–BB	lin–OA	qua–OA	LP/NLP	QP/NLP
TP1	0.49	0.60	0.59	0.41	0.39
TP2	1.77	2.17	2.27	1.22	1.71
TP3	7.27	12.51	13.92	4.56	6.25
BATCH	151.33	58.27		26.46	57.38
BATCH–int	25.67	9.73	13.06	8.47	9.67
Asaadi 1 (3)	0.22	0.21	0.14	0.15	0.14
Asaadi 1 (4)	0.64	0.48	0.46	0.39	0.29
Asaadi 2 (4)	0.87	11.21	4.76	3.08	1.83
Asaadi 2 (7)	1.03		6.10	4.61	1.74
Asaadi 3 (6)	4.21	57.93	28.58	17.38	9.41
Asaadi 3 (10)	17.59	195.89	96.69	48.84	33.38
2DEx	0.09	0.17	0.10	0.09	0.09
GTD	1.50	2.55		2.62	0.65
GTD chain(2)	0.62	0.18	0.24	0.15	0.16
GTD chain(3)	2.94	1.49		0.78	0.41
AVGAS 1	0.70	1.42	3.05	0.67	0.72
AVGAS 2	0.91	1.53	3.04	0.53	0.77
AFIRO (3)	6.18	12.32	20.30	10.06	5.66
AFIRO (5)	36.55	83.54	50.63	30.14	17.42
AFIRO (6)	51.73	265.44	159.03	43.75	39.75
AFIRO (7)	33.04	150.04	96.94	37.19	29.25

Table 6.5: CPU times for solves

On the first class of problems, the best routines are the linear outer approximation routines, especially the LP/NLP based branch–and–bound routine. This routine employs "cheap" LP problems to search the tree and solves only a small number of NLP problems. Since the functions are linear in y and due to the fact that any nonlinear function over $\{0,1\}^n$ that is separable can be replaced by a linear function, the linear model seems most appropriate and second order information seems to be wasted. Moreover, quadratic information can be misleading, if the function concerned is a monotone function and such an example has been presented in Chapter 5.

On the second class of test problems, the linear outer approximation routines are outperformed by either nonlinear branch–and–bound or by the quadratic outer approximation routines. This appears to be due to the fact that the nonlinear functions are not adequately represented by linearizations. If the integer solution can be obtained by rounding the solution of the NLP relaxation of the MINLP test problem then nonlinear branch–and–bound appears to be the best choice. This situation corresponds to inequality constrained problems. In this case the *integrality gap*, that is the difference between the objective value of the NLP relaxation and the objective value of the integer solution, is small. If, however, the evaluation of the objective function becomes very costly, as in GTD, then it is cheaper to search the tree using the less accurate quadratic model rather than the more costly nonlinear functions.

A class of problems whose solution cannot be obtained by rounding is formed by problems

with linear equality constraints. These problems are likely to have a large integrality gap and branch–and–bound is less efficient than the QP/NLP based branch–and–bound routine, since the tree search becomes more expensive. The AFIRO problems fall into this class and indicate that it can be advantageous to use outer approximation routines with quadratic information for the solution of difficult MINLP problems.

The experience that has been gained by running the test problems can be summarized to give a "Rule of Thumb" which acts as a guide as to when which routine can be expected to produce best results.

Rule of Thumb for usage of MINLP routines

Method	Ideal Situation
LP/NLP	Binary MINLP problems.
nonlin–BB	General MINLP with a small integrality gap.
QP/NLP	General MINLP with a large integrality gap.

Chapter 7

Nonsmooth MINLP

7.1 Introduction

The aim of this chapter is to generalize the outer approximation algorithms of the previous chapter to cover nonsmooth MINLP problems. Apart from being of interest in their own right , nonsmooth MINLP problems arise naturally through exact penalty function formulations of smooth MINLP problems like P. This type of problem is clearly not covered by the model problem P so that new techniques are required.

Throughout the previous chapters it has been assumed that problem P is a smooth MINLP problem. A careful analysis of the derivation of the master program and the algorithm reveals two reasons for this assumption. The first reason is that f and g are required to be smooth so that their supporting hyperplanes are well defined and outer approximations can readily be derived. The second reason is that the Kuhn–Tucker optimality conditions assume differentiability.

In order to derive a master program and hence outer approximation algorithms that can handle non–differentiable problems it is necessary to replace the two features that require differentiability. The outer approximation by a supporting hyperplane becomes consequently the outer approximation by a subdifferential and the Kuhn–Tucker conditions are replaced by suitable first order conditions for nonsmooth NLP (e.g. [17], p. 406). Figure 2.1 of Chapter 2 shows the outer approximation of a convex nonsmooth function.

The most common occurrence of a nonsmooth MINLP problem is through an exact penalty function formulation of the model problem P. In this case, the function to be minimized has a smooth part, the original objective function, and a nonsmooth penalty part, derived from the violation of the nonlinear constraints. The class of problems considered here covers all common exact penalty function formulations and some other nonsmooth functions. The model problem for this chapter is

$$\mathrm{P}_{ns} \left\{ \begin{array}{ll} \min\limits_{x,y} & f(x,y) + h(g(x,y)) \\ \text{subject to} & x \in X \ , \ y \in Y \text{ integer} \end{array} \right.$$

where f and g are continuously differentiable and X is as in assumption **A1**. It is assumed that $h : \mathbb{R}^m \rightarrow \mathbb{R}$ is convex but nonsmooth. This assumption alone does not imply convexity of P_{ns} which is needed to enable its treatment by outer approximation. It is therefore convenient to assume that h is also a monotone function, that is

$$a \leq b \ \Rightarrow \ h(a) \leq h(b).$$

And it follows that P_{ns} is a convex programming problem provided X, f and g are convex. In many cases $h(g)$ is a polyhedral convex function such as $h(g) = \max_i g_i$, $h(g) = \|g^+\|_\infty$ or $h(g) = \|g^+\|_1$,

but other functions are also possible. (Here a^+ denotes the vector $a^+ = (a_1^+, \ldots, a_m^+)^T$ where $a_i^+ = \max(a_i, 0)$.)

Similarly to Chapters 4 and 5, problem P_{ns} can be reformulated using projection and first order conditions to obtain a (nonsmooth) master problem. This master problem is equivalent to an MILP problem if extra variables are added. Relaxations of this master program are used in deriving an outer approximation algorithm which iterates finitely between nonsmooth NLP subproblems and MILP master program relaxations. Exact penalty functions form a subclass of the type of optimization problems considered here and their use in outer approximation is further examined. The main attraction of exact penalty functions lies in the fact that they make a distinction between feasible and infeasible subproblems unnecessary. Sufficient conditions are examined under which the standard MINLP problem and its exact penalty function formulation are equivalent.

The chapter is divided into two main sections. In Section 7.2 the master program for P_{ns} is derived and the equivalence of P and its exact penalty function formulation is examined. Particular attention is given to the l_1 exact penalty problem and it is shown in this case how extra variables can be used to convert the nonsmooth master problem to an MILP problem. Section 7.3 derives the outer approximation algorithms and presents the finite convergence proof. An alternative version of outer–approximation is developed, where only one cut is added to the master program relaxations per iteration and it is shown that – under certain conditions – this cut is equivalent to the corresponding Benders cut.

7.2 Reformulation of nonsmooth MINLP problems

The reformulation of P_{ns} employs the same techniques as those used in Chapter 4 to reformulate P. First a projection of P_{ns} onto the integer variables is defined, which gives rise to a nonsmooth NLP subproblem. The subgradient inequality and first order necessary conditions imply that this subproblem can be replaced by a nonsmooth but linear subproblem. Finally, the convexity assumption and the monotonicity of h ensure that the linearizations of the subproblems are supporting hyperplanes of P_{ns} and the master program is obtained.

First, the projection onto the integer variables

$$\mathrm{proj}(\mathrm{P}_{ns}) \left\{ \min_{y^j \in Y} \{\mathrm{NSO}(y^j)\}. \right.$$

is defined, where the nonsmooth subproblem $\mathrm{NSO}(y^j)$ is obtained from P_{ns} by fixing the integer variables at $y = y^j$, that is

$$\mathrm{NSO}(y^j) \left\{ \begin{array}{ll} \min\limits_{x} & f(x, y^j) + h(g(x, y^j)) \\ \text{subject to} & x \in X. \end{array} \right.$$

Let x^j be an optimal solution of $\mathrm{NSO}(y^j)$. As a consequence of the subgradient inequality and the first order necessary conditions (Theorem 2.5.1) $\mathrm{NSO}(y^j)$ has the same solution as the following linearized problem.

$$\left\{ \begin{array}{ll} \min\limits_{x,\eta} & \eta \\ \text{subject to} & \eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ 0 \end{pmatrix} + h(g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ 0 \end{pmatrix}) \\ & x \in X \end{array} \right. .$$

where a dummy variable η has been introduced. Let $\eta^j = f^j + h(g^j)$ denote the optimal value of η. Replacing $\mathrm{NSO}(y^j)$ by its linearization implies that the projected problem $\mathrm{proj}(\mathrm{P}_{ns})$ has the same

solution as

$$\min_{y^j \in Y} \left\{ \begin{array}{lll} \min\limits_{x,\eta} & \eta & \\ \text{subject to} & \eta \geq f^j + (\nabla f^j)^T \left(\begin{array}{c} x - x^j \\ 0 \end{array} \right) + h(g^j + [\nabla g^j]^T \left(\begin{array}{c} x - x^j \\ 0 \end{array} \right)) \\ & x \in X \end{array} \right\}.$$

Next it is shown that the two min operations in the above projection can be combined into one single master program

$$\mathrm{M}_{ns} \left\{ \begin{array}{lll} \min\limits_{\eta,x,y} & \eta & \\ \text{subject to} & \eta \geq f^j + (\nabla f^j)^T \left(\begin{array}{c} x - x^j \\ y - y^j \end{array} \right) + h(g^j + [\nabla g^j]^T \left(\begin{array}{c} x - x^j \\ y - y^j \end{array} \right)) & j \in T \\ & x \in X \ , \ y \in Y \text{ integer} \end{array} \right.$$

where $T = \{j : x^j \text{ is an optimal solution to NSO}(y^j)\}$. It suffices to show that (η^i, x^i, y^i) is feasible in M_{ns} for all $i \in T$. To prove this consider

$$\eta^i = f^i + h(g^i).$$

The convexity of f ensures that a first order expansion of f about (x^j, y^j) underestimates f^i, so that

$$\eta^i \geq f^j + (\nabla f^j)^T \left(\begin{array}{c} x^i - x^j \\ y^i - y^j \end{array} \right) + h(g^i).$$

Finally the convexity of g and the monotonicity of h imply that

$$\eta^i \geq f^j + (\nabla f^j)^T \left(\begin{array}{c} x^i - x^j \\ y^i - y^j \end{array} \right) + h(g^j + [\nabla g^j]^T \left(\begin{array}{c} x^i - x^j \\ y^i - y^j \end{array} \right)).$$

so that (η^i, x^i, y^i) is feasible in M_{ns}. The above development provides a proof of the following Theorem.

Theorem 7.2.1 *If assumptions* **A1** *and* **A2** *hold, then M_{ns} is equivalent to P_{ns} in the sense that (η^*, x^*, y^*) solves P_{ns} if and only if it solves M_{ns}.*

The master program M_{ns} is not solved directly but instead a relaxation strategy similar to the algorithms of Chapter 5 is applied and this is explained in Section 7.3. The Theorem 7.2.1, unlike its equivalent in Chapter 4 does not require a constraint qualification to hold. The reason for this is that the first order necessary conditions that are used in reformulating P_{ns} do not require a constraint qualification.

A class of nonsmooth MINLP which is of particular interest are exact penalty functions. Exact penalty functions offer an alternative approach to the difficulties caused by infeasible subproblems and make a distinction between feasible and infeasible NLP subproblems unnecessary. Instead of solving problem P an exact penalty function formulation of P is considered.

$$\mathrm{E} \left\{ \begin{array}{ll} \min\limits_{x,y} & \Phi(x,y) = \nu f(x,y) + \|g(x,y)^+\| \\ \text{subject to} & x \in X, \ y \in Y \text{ integer.} \end{array} \right.$$

where $\| \cdot \|$ is a norm in $\mathbb{R}^m$, and ν is a sufficiently small penalty parameter. This is a special case of P_{ns} and the above development applies.

It is of interest to know under which conditions E and P are equivalent. This is an important question since it indicates when an algorithm based on the master program M_{ns} solves the original problem P. The next theorem gives sufficient conditions under which the mixed integer exact penalty function problem E is equivalent to problem P, so that any algorithm based on solving relaxations of M_{ns} terminates at a solution to P. One of these conditions is that the penalty parameter has to be "sufficiently small". This is qualified by the following conditions on the penalty parameter, where $\|\cdot\|_D$ denotes the dual norm to $\|\cdot\|$ introduced in Chapter 2.

A5 Let the penalty parameter ν satisfy

$$\nu < \frac{1}{\max_j \|\lambda^j\|_D} \qquad \forall j: \text{ NLP}(y^j) \text{ is feasible}$$

$$\nu < \frac{\|(g^k)^+\|}{f^* - f^k} \qquad \forall k: \|(g^k)^+\| > 0 \text{ and } f^k < f^*.$$

A6 Let a second order sufficient condition (e.g. Theorem 2.5.2) hold for all j such that NLP(y^j) is feasible.

Although additional assumptions have to be made, **A6** will usually hold. If the user's choice of the penalty parameter does not satisfy **A5** then the optimal solution of E is not feasible in P. The user can detect this fact and reduce the penalty parameter accordingly.

The first condition in **A5**, together with **A6** is needed to ensure that the solution of the feasible NLP–subproblems and the corresponding exact penalty function–subproblems are equivalent, and the second condition in **A5** ensures that any outer approximation algorithm does not terminate with an infeasible solution. A simple conclusion of Theorem 7.2.2 is that any outer approximation algorithm terminates finitely at a solution of P or, if P is infeasible, it finds the "best" exact penalty solution to P. Now Theorem 7.2.2 can be stated

Theorem 7.2.2 *If assumptions* **A1** *to* **A6** *hold and if P has a feasible solution, then E and P are equivalent in the sense that* (x^*, y^*) *solves P if and only if it solves E.*

Proof:
Assumptions **A3**, **A6** and the first part of assumption **A5** imply that any feasible NLP(y^j) subproblem of P is equivalent to the corresponding NSO(y^j) subproblem of E (c.f. [17], Theorem 14.3.1, p. 380). It remains, therefore, to show that the solution of E cannot be a point (x^k, y^k) for which NLP(y^k) is infeasible. Now let (x^k, y^k) be such that $\|(g^k)^+\| > 0$. The second part of assumption **A5** implies that

$$\begin{aligned} \Phi^k &= \nu f^k + \|(g^k)^+\| \\ &> \nu f^k + \nu(f^* - f^k) \\ &= \nu f^* \end{aligned}$$

Therefore, $\Phi^k > \Phi^*$ which concludes the proof.
□

If $h(g)$ is a polyhedral convex function, it is possible to reformulate the constraints in M_{ns} using a standard linear programming technique. If $h(g) = \|g^+\|_1$, then additional variables ξ_l are introduced and the constraints are equivalent to

$$\left.\begin{aligned} \eta &\geq f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} + \sum_{l=1}^{m} \xi_l \\ \xi_l &\geq g_l^j + (\nabla g_l^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} && l = 1, \ldots, m \\ \xi_l &\geq 0 && l = 1, \ldots, m \end{aligned}\right\} (C_1)$$

In the case of $h(g) = \|g^+\|_\infty$ only a single additional variable is needed.

An alternative way of deriving the constraints C_1 is now explained in the context of the l_1 exact penalty function problem. It is possible to introduce variables ξ_l directly into E so that it can be reformulated as

$$\mathrm{E}_1 \left\{ \begin{array}{lll} \min\limits_{x,y,\sigma} & \nu f(x,y) + \sum\limits_{l=1}^{m} \xi_l & \\ \text{subject to} & \xi_l \geq g_l(x,y) & l = 1, \ldots, m \\ & \xi_l \geq 0, \ \forall l, \ x \in X, \ y \in Y \text{ integer}, & \end{array} \right.$$

Outer approximations of E_1 can be derived using the methods of Chapter 4, giving rise to the constraints C_1 (with f replaced by νf). A similar formulation is again possible for the l_∞ norm.

7.3 Outer Approximation Algorithms for nonsmooth MINLP problems

The outer approximation algorithms based upon M_{ns} work in a similar fashion to those proposed in Chapter 5. This section, therefore, restricts attention to the derivation of a linear outer approximation algorithm for nonsmooth problems, although an indication is also given how the other algorithm can be generalized.

Like the master program M of Chapter 4 it is not practical to solve M_{ns} directly. Instead relaxations of M_{ns} are used in an iterative procedure. The relaxation M^i_{ns} that is solved at iteration i of the algorithm is

$$M^i_{ns} \left\{ \begin{array}{lll} \min\limits_{\eta,x,y} & \eta & \\ \text{subject to} & \eta < \mathrm{UBD}^i & \\ & \eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} + h(g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix}) & j \in T \\ & x \in X \ , \ y \in Y \text{ integer}. & \end{array} \right.$$

where $T^i = \{j \leq i : x^j \text{ is an optimal solution to NSO}(y^j)\} \subset T$ and

$$\mathrm{UBD}^i = \min_{j \leq i} \{f^j + h(g^j)\}.$$

The program M^i_{ns} can now be used in an outer approximation algorithm similar to Algorithm 1. The only unusual feature is the occurrence of the convex composition $h(g)$. However, using standard linear programming techniques, $h(g)$ can be expressed as a set of linear inequality constraints if h is a polyhedral function, and this is described in Section 7.2 for the l_1 norm. The new algorithm can now be described as follows.

Algorithm 6: Nonsmooth Outer Approximation

Initialization: y^0 is given; set $i = 0$, $T^{-1} = \emptyset$ and $\mathrm{UBD}^{-1} = \infty$.

REPEAT

1. Solve the subproblem $\mathrm{NSO}(y^i)$ and let the solution be x^i.

2. Linearize the objective and (active) constraint functions about (x^i, y^i).
   Set $T^i = T^{i-1} \cup \{i\}$.

3. **IF** $(f^i + h(g^i) < \mathrm{UBD}^{i-1})$ **THEN**
   update current best point by setting $x^* = x^i,\ y^* = y^i,\ \mathrm{UBD}^i = f^i + h(g^i)$.
   ELSE
   Set $\mathrm{UBD}^i = \mathrm{UBD}^{i-1}$.

4. Solve the current relaxation M^i_{ns} of the master program M_{ns}, giving a new integer assignment y^{i+1} to be tested in the algorithm. Set $i = i + 1$.

UNTIL (M^i_{ns} is infeasible).

The following theorem establishes the finite convergence of the algorithm.

Theorem 7.3.1 *If assumptions* **A1** *and* **A2** *are satisfied and Y is finite then Algorithm 6 converges finitely to a solution of P_{ns}.*

Proof:
It is shown first that no integer assignment is generated twice by the algorithm. Its finiteness then follows from the finiteness of Y.

It is assumed that at iteration $i \geq j$ the integer assignment y^j is feasible in the master program M^i_{ns} and a contradiction is sought. It follows that there exists an $x' \in X$ satisfying the inequality

$$\eta \geq f^j + (\nabla f^j)^T \begin{pmatrix} x' - x^j \\ 0 \end{pmatrix} + h(g^j + [\nabla g^j]^T \begin{pmatrix} x' - x^j \\ 0 \end{pmatrix}).$$

(The assumption $X \neq \emptyset$ ensures the existence of a solution x^j to $\mathrm{NSO}(y^j)$.) Let $\lambda^j \in \partial h(g^j)$ be the optimal multiplier of $\mathrm{NSO}(y^j)$. It follows from the definition of the subdifferential $\partial h(g)$ that

$$\eta \geq f^j + h(g^j) + (\nabla f^j + \nabla g^j \lambda^j)^T \begin{pmatrix} x' - x^j \\ 0 \end{pmatrix}. \tag{7.1}$$

In order to apply the optimality conditions of Theorem 2.5.1 it is convenient to handle the constraint $x' \in X$ by introducing composite functions. Since X contains only linear functions like $r_i(x) = r_i^T x - b_i \leq 0,\ i = 1, \ldots, q$ these constraints can be fitted into the framework of the above optimality conditions through the single constraint

$$t(r(x)) \leq 0$$

involving the polyhedral function

$$t(r(x)) = \max_i r_i(x).$$

The optimality of x^j implies the existence of multipliers $\pi^j \geq 0$ and $\mu^j \in \partial t(r^j)$. Premultiplying the linear constraint by $\pi^j \mu^j_i$, summing over all $i = 1, \ldots, q$ and adding to (7.1) gives the following valid inequality

$$\eta \geq f^j + h(g^j) + t(r^j)\pi^j + (\nabla f^j + \nabla g^j \lambda^j + \nabla r^j \mu^j)^T \begin{pmatrix} x' - x^j \\ 0 \end{pmatrix}.$$

The first order necessary conditions for NSO(y^j) (e.g. Theorem 2.5.1) imply that

$$\begin{array}{ll} t(r^j)\pi^j = 0 & \textit{complementarity} \\ \nabla f^j + \nabla g^j \lambda^j + \nabla r^j \mu^j = 0 & 1^{st}\ \textit{order condition.} \end{array}$$

Thus the inequality

$$\eta \geq f^j + h(g^j)$$

can be derived for η. This contradicts the strict upper bound on η which is

$$\eta < \ \text{UBD}^i \leq f^j + h(g^j).$$

This concludes the argument to show that Algorithm 6 is finite. Next the convergence to an optimal solution is established. Assume that Algorithm 6 terminates with an indicated solution for which

$$\text{UBD}^i = f' + h' > f^* + h(g^*).$$

The convexity assumption implies that y^* must be feasible in the previous MILP master program relaxation which contradicts the termination assumption and concludes the proof.
□

It is worth mentioning, that Algorithm 6 does not require a constraint qualification on g to hold in order to achieve finite convergence. However, such an assumption is needed to show that the exact penalty function formulation of P and P itself are equivalent as is shown in Section 7.2.

The proof of Theorem 7.3.1 indicates that it is possible to derive a version of Algorithm 6 in which only one constraint is added per iteration. This single cut is given by

$$\eta \geq f^j + h(g^j) + (\nabla f^j + [\nabla g^j]\lambda^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix}$$

where $\lambda^j \in \partial h(g^j)$ is the optimal multiplier vector of the NSO(y^j) subproblem. It is instructive to compare this cut to the Benders cut for the same problem. It is shown in Chapter 4 that the Benders cut can be written as

$$\eta \geq f^j + h(g^j) + \mu^j(y^j - y)$$

where μ^j is the optimal multiplier of the constraint $y = y^j$ in

$$NSO'(y^j) \left\{ \begin{array}{ll} \min\limits_{x,y} & f(x,y) + h(g(x,y)) \\ \text{subject to} & y = y^j \\ & x \in X\ ,\ y \in Y \end{array} \right.$$

The first order necessary conditions ([17], Theorem 14.6.1, p. 406 f.) enable an expression of μ^j in terms of ∇f^j, ∇g^j, and $\lambda^j \in \partial h(g^j)$ to be given

$$\nabla_y f^j + [\nabla_y g^j]\lambda^j + \mu^j = 0$$

so that the Benders cut can finally be written as

$$\eta \geq f^j + h(g^j) + (\nabla_y f^j + [\nabla_y g^j]\lambda^j)^T(y - y^j).$$

Clearly, if $x^j \in X$ lies in the strict interior of X (or if all corresponding multipliers μ^j are zero), then also $\nabla_x f^j + [\nabla_x g^j]\lambda^j = 0$ and both cuts are equivalent. This last statement indicates that it might not be advisable to use just the single cut, since Benders Decomposition is usually inferior to outer approximation.

In Chapter 5 it is shown that linear outer approximation cannot be expected to work well for problems in which curvature plays a prominent part. The same observation applies for the generalization to nonsmooth MINLP problems presented in this chapter. Hence, it is now indicated how the other three algorithms of Chapter 5 should be modified in order to be applicable to P_{ns}.

The definition of the set S^i becomes obsolete and all algorithms use relaxations M^i_{ns} or Q^i_{ns} where Q^i_{ns} is obtained from M^i_{ns} by adding the quadratic term

$$\tfrac{1}{2}\begin{pmatrix} x-x^i \\ y-y^i \end{pmatrix}^T \nabla^2\mathcal{L}^i \begin{pmatrix} x-x^i \\ y-y^i \end{pmatrix}$$

to the objective function of M^i_{ns}, where $\mathcal{L} = f + \lambda^T g$. Instead of solving NLP–subproblems the new algorithms solve NSO-subproblems. The condition "NLP(y^i) feasible" becomes redundant in the IF–statement of Step 3 and Step 2(d) respectively, so that the current best point is updated whenever $f^i + h(g^i) < \mathrm{UBD}^{i-1}$ is satisfied.

Finally it is possible to generalize problem P_{ns} even further by including a composite constraint of the form

$$t(r(x,y)) \leq 0.$$

The nonsmooth problem consequently becomes

$$\mathrm{P}_{ns}\begin{cases} \min\limits_{x,y} & f(x,y) + h(g(x,y)) \\ \text{subject to} & t(r(x,y)) \leq 0 \\ & y \in Y \text{ integer} \end{cases}$$

where the constraints $x \in X$ are now included in the constraints $t(r(x,y)) \leq 0$ as indicated in the proof of Theorem 7.3.1. Fletcher [17] gives first order necessary conditions for the NSO subproblem obtained from this class of problem.

It is possible to derive an equivalent MILP master program using similar techniques to those employed in this section. The inclusion of this additional constraint has, however, the disadvantage that it makes a separate treatment of infeasible subproblems necessary, whereas the main reason for introducing penalty functions is that this is avoided.

Chapter 8

Nonconvex and Inexact MINLP

8.1 Introduction

In practice, the convexity assumption **A1** is often too restrictive, since many practical applications involve nonlinear equality constraints or nonconvex objectives or constraints. There have been various heuristic attempts ([42], [73]) to modify the linear outer approximation algorithm of Duran and Grossmann in order to make it more suitable for nonconvex MINLP problems. This chapter presents a deterministic outer approximation algorithm for solving a class of nonconvex MINLP problems. Based on this algorithm a new heuristic is proposed for solving more general nonconvex MINLP problems by outer approximation.

In practical implementations it is usually not possible to satisfy assumption **A4**, since errors such as round–off errors or inexact solutions to NLP problems cannot be avoided. It is shown in this chapter how outer approximation algorithms can take those errors into account and it is suggested that the tolerance ϵ that is used in the upper bound $\eta \leq \text{UBD} - \epsilon$ should be determined by the algorithms depending on estimates of the various errors encountered during the solution of the MINLP problem.

Both nonconvex and inexact MINLP problems can be treated by the same framework algorithm. The central idea of this algorithm is the introduction of tolerances associated with the "quality" of the outer approximations, the NLP solvers and the MILP/MIQP solvers. The new algorithm is shown to converge finitely to an ϵ optimal solution. While the use of tolerances provides a rigorous way of determining the overall accuracy of the outer approximation algorithms they also introduce additional flexibility into outer approximation which can be exploited for nonconvex problems.

In order to solve nonconvex MINLP problems the idea of linear underestimators is introduced and combined with the outer approximation concept to give an algorithm which minimizes a concave pure integer function subject to a set of convex nonlinear constraints. It is also indicated how heuristics can be developed for MINLP problems with nonconvex objective functions and convex constraints.

The aim of this chapter is to point towards interesting new developments in the area of outer approximation that might profitably be explored in the future. The chapter is divided into five parts. The next section gives an overview over some approaches developed to handle nonconvexities in the model problem via outer approximation. In Section 8.3 a general outer approximation framework is presented and finite convergence to an ϵ–optimal solution is proved. Sections 8.4 and 8.5 consider two applications of the new framework to nonconvex MINLP problems and to inexact NLP and MILP/MIQP solvers.

8.2 Heuristic methods for nonconvex MINLP

In considering nonconvex MINLP problems it is useful to be aware that there exist classes of problems which are not solvable by any algorithm. Jeroslow [39] shows that the minimum of a nonconvex quadratically constrained integer problem is not computable by a recursive function. Since every routine programmable on a computer is a recursive function, this implies that there are no means of devising an algorithm (not even a very inefficient one) to solve this class of problem on a computer. The proof of this fundamental statement uses a result on diophantine equations which provides a solution to the tenth Hilbert problem. An important class of problem that *is* solvable is the class of integer problems with a bounded feasible region. There have been several attempts to devise good heuristics for the outer approximation algorithms of Chapter 5 and these are discussed in the remainder of this section.

It is sometimes possible to reformulate the problem avoiding the nonconvexities. Torres [70] gives a reformulation that transforms nonconvex constraints of the form

$$y_i F(x) + h(x, y) \leq 0,$$

where $y_i \in \{0, 1\}$, into a set of convex constraints, provided both h and F are convex functions. The mixed product $y_i F(x)$ is replaced by a new continuous variable p and the following constraints are added to the problem.

$$\begin{aligned} p &\geq F(x) - U(1 - y_i) \\ p &\geq L y_i, \end{aligned}$$

where $L < F(x)$ and $U > F(x)$ for all feasible $x \in X$. In doing so he improves on an earlier reformulation by Glover [29] which did not result in a set of convex constraints. Kocis and Grossmann [42] give a logarithmic transformation that convexifies certain nonconvex functions.

A two phase heuristic strategy to solve nonconvex MINLP problems is proposed by Kocis and Grossmann [42]. In the first phase outer approximation is applied to the problem without regard for its nonconvexities. Upon termination of phase I the current incumbent provides an upper bound on the optimum but the linearizations are not necessarily outer approximations. Kocis and Grossmann suggest to apply *local* and *global* tests to the linearizations contained in the last master program to determine whether any linearizations cut into the feasible region. If all linearizations pass these tests then it is assumed that the solution is optimal. Otherwise slack variables are added to those linearizations that fail a test and penalty terms involving the slack variables are added to the master program which is then solved in phase II. All linearizations added to the master problem in phase II are checked for convexity when they are introduced into the master problem.

The *local* test consists of solving the following perturbed NLP for every integer assignment y^i that is generated in phase I.

$$\begin{cases} \min\limits_{x,y} & f(x, y) \\ \text{subject to} & g(x, y) \leq 0 \\ & x \in X,\ y \in Y \\ & \|x - x^i\|_\infty \leq \epsilon \\ & \|y - y^i\|_\infty \leq \epsilon, \end{cases}$$

where ϵ is small, typically 0.005. The aim of this test is to relax the integrality condition $y = y^i$ so that the neighbourhood of the point (x^i, y^i) can be explored. Good starting points are available from phase I of outer approximation and the local test then checks that the linearizations of f and g about (x^i, y^i) are also valid for the solution of the above NLP problem. The *global* tests consist of checking that all integer assignments generated in phase I satisfy all linearizations. It is clear that the test might fail to recognize nonconvexities and a simple example where this occurs is given in Figure 8.1. The linearizations about $y^0 = 0$ satisfy both the local and the global tests.

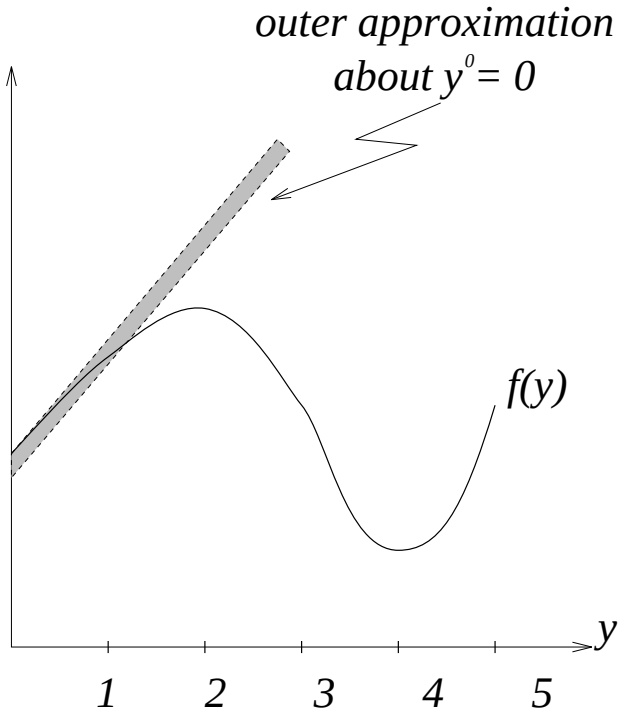

Figure 8.1: Example of a nonconvex MINLP

A more fundamental difficulty with the approach suggested by Kocis and Grossmann is that by adding slack variables to the linearizations of phase I which fail the test, the corresponding integer assignment can be made feasible again in the master program. Kocis and Grossmann therefore use integer cuts to avoid regenerating the same integer assignment again. These integer cuts, however, are only readily available for binary variables and the dilemma of relaxing the linearizations to allow for nonconvexities whilst maintaining the finite convergence properties of outer approximation is not resolved.

An alternative approach is proposed by Viswanathan and Grossmann [73] based on an augmented penalty formulation. Instead of solving the master program M^i at each iteration of outer approximation they suggest to solve an MILP where nonnegative slack variables are added to the master, that is

$$
M'^i \left\{ \begin{array}{lll}
\min\limits_{x,y,\eta} & \eta + \sum\limits_{j \le i} w_0^j s_0^j + \sum\limits_{l,j \le i} w_l^j p_l^j & \\
\text{subject to} & \eta + s_0^j \ge f^j + (\nabla f^j)^T \left(\begin{array}{c} x - x^j \\ y - y^j \end{array} \right) & \\
 & & \forall j \in T^i \\
 & p^j \ge g^j + [\nabla g^j]^T \left(\begin{array}{c} x - x^j \\ y - y^j \end{array} \right) & \\
 & p^k \ge g^k + [\nabla g^k]^T \left(\begin{array}{c} x - x^k \\ y - y^k \end{array} \right) & \forall k \in S^i \\
 & x \in X,\ y \in Y \text{ integer} & \\
 & s_0^j \ge 0\ ,\ p^j \ge 0\ ,\ p^k \ge 0 &
\end{array} \right.
$$

where w_0^j and w_l^j are nonnegative weights on the slacks.

Finite termination can only be guaranteed if integer cuts are used which makes this approach unsuitable for general MINLP problems. Moreover, not only does the number of constraints in the master program grow with every iteration but also the number of variables. Viswanathan and Grossmann circumvent the finite termination problem by proposing a new heuristic stopping rule. Outer approximation is stopped whenever $f^{\text{new}} > f^{\text{old}}$, where f^{old} is the incumbent value and f^{new} is the newest value of the objective function. It is not difficult to find a *convex* MINLP where this test results in a premature stop at a suboptimal solution. The worst case example of Chapter 5 provides such an example. In the first step $f(1) > f(0)$ and the outer approximation stops at the incorrect minimum. As a consequence, the convex MINLP case is *not* recovered as a special case of Viswanathan and Grossmann's routine.

Both algorithms have difficulties since they simply *shift* the linearizations thus removing the problem of cutting into the feasible region at the expense of a finite convergence property. The approach to nonconvex MINLP presented in Section 8.4 redresses this dilemma by replacing linearizations by more flexible linear underestimators.

8.3 A framework outer approximation algorithm for nonconvex and inexact MINLP problems

In this section a general framework outer approximation algorithm is presented that is applicable to a class of nonconvex MINLP problems and which also addresses the problem of inexact solutions to the NLP subproblems and the MILP/MIQP master programs. The main idea is the introduction of tolerances measuring the accuracy of the linear supporting hyperplane, and of the NLP and MILP solutions, similar to a recent idea for Benders' Decomposition by Flippo and Rinnoy Kan [22]. The finite termination proof of Chapter 5 is restated for the new generalized procedure. Section 8.4 and 8.5 show how the new scheme can be adapted to handle nonconvex MINLP problems and inexact solutions to NLP and MILP problems.

The convexity assumption **A1** of Chapter 1 is replaced here by the following weaker assumption, denoted by **A1'**.

A1' X is a nonempty compact convex set defined by a system of linear inequality constraints, the constraint functions g are convex in x and y jointly and there exist linear underestimators for the objective function f given by

$$f(x,y) \geq c^i + (a^i)^T \begin{pmatrix} x - x^i \\ y - y^i \end{pmatrix} \ \forall (x,y) \in X \times Y$$

and there exist multipliers $u^i \geq 0$ such that $a_x^i = -[\nabla_x g^i]^T u^i$ and $u_j^i = 0, \ \forall j \notin \mathcal{A}^i$.

The existence of multipliers implies that for $y = y^i$ no linearized feasible descent direction exists in the master program, which is important in the finite convergence proof. Any convex f clearly satisfies assumption **A1'** (with $c^i = f^i$ and $a^i = \nabla f^i$), but there also exist important classes of nonconvex problems which satisfy it such as problems involving pure integer concave minimization over a convex set.

The algorithm introduces tolerances $\epsilon_1^i, \epsilon_2^i$ and ϵ_3^i which measure the accuracy with which the NLP(y^i) subproblems are solved, the tightness of the linear support at the point (x^i, y^i) and the accuracy with which the MILP master program relaxations are solved. The overall accuracy ϵ_0^i depends on these tolerances and consequently cannot be given a priori, but is modified at each

iteration. Instead of using linearizations of the objective function f it is assumed that linear underestimators can be computed. This assumption replaces the stronger convexity assumption in the next section and allows a wider class of MINLP problems to be solved by outer approximation. The algorithm is now stated in full detail and provides only a small modification to Algorithm 1 of Chapter 5.

Algorithm 7: Outer Approximation Framework

Initialization: y^0 is given; set $i = 0$, $T^{-1} = \emptyset$, $S^{-1} = \emptyset$, $\text{UBD}^{-1} = \infty$ and $\text{LBD}^{-1} = -\infty$.

REPEAT

1. Obtain an ϵ_1^i optimal solution to the subproblem NLP(y^i), or a feasibility problem F(y^i) if NLP(y^i) is infeasible, and let the solution be x^i.

2. Linearize the (active) constraint functions about (x^i, y^i).
   Obtain a linear underestimator of f, that is find
   $$\eta \geq c^i + (a^i)^T \begin{pmatrix} x - x^i \\ y - y^i \end{pmatrix}$$
   Set $\epsilon_2^i \geq f^i - c^i \geq 0$.
   Set $T^i = T^{i-1} \cup \{i\}$ or $S^i = S^{i-1} \cup \{i\}$ as appropriate.

3. **IF** (NLP(y^i) is feasible and $f(x^i, y^i) < \text{UBD}^{i-1}$) **THEN**
   update current best point by setting $x^* = x^i,\ y^* = y^i,\ \text{UBD}^i = f(x^i, y^i)$.
   ELSE
   Set $\text{UBD}^i = \text{UBD}^{i-1}$.

4. Set $\epsilon_0^i > \epsilon_1^i + \epsilon_2^i + \epsilon_3^i$, where ϵ_3^i is the accuracy of the MILP solver and solve the current relaxation M'^i of the master program,
   $$M'^i \begin{cases} \min\limits_{x,y,\eta} & \eta \\ \text{subject to} & \eta \leq \text{UBD}^i - \epsilon_0^i \\ & \eta \geq c^j + (a^j)^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} \\ & \qquad\qquad \forall j \in T^i \\ & 0 \geq g^j + [\nabla g^j]^T \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} \\ & \\ & 0 \geq g^k + [\nabla g^k]^T \begin{pmatrix} x - x^k \\ y - y^k \end{pmatrix} \quad \forall k \in S^i \\ & x \in X,\ y \in Y \text{ integer} \end{cases}$$
   giving a new integer assignment y^{i+1} to be tested in the algorithm. Set $i = i + 1$.

5. A lower bound is given by $\text{LBD}^i = \max(\text{LBD}^{i-1}, \eta^i - \epsilon_3^{i-1})$.

UNTIL (M'^i is infeasible).

Algorithm 7 differs in two important aspects from the linear outer approximation Algorithm 1. The use of linear underestimators gives a greater flexibility when dealing with nonconvex problems

and includes convex MINLP problems as a special case, since linear underestimators of convex functions are readily available through linearizations of f. The use of tolerances remedies another problem of outer approximation that can sometimes be observed in practice. While exact NLP/MILP/MIQP solvers guarantee that no integer assignment is generated twice, the same holds for inexact solvers *only if* ϵ *is "sufficiently large"* and this notion is rigorously qualified in Section 8.5. It is also argued in Section 8.5 that integer cuts provide a strong non–repetition guarantee, but these are only available for binary variables.

The fact that the tolerance ϵ_0^i is dependent on i means that it may decrease as well as increase from iteration to iteration. Consequently, it is only possible to show that no complete master program solution (η^i, y^i) is generated twice by the algorithm, whereas an integer assignment y^i might be generated again for a smaller value of ϵ_0. The following Lemma states that no complete solution is generated twice.

Lemma 8.3.1 *If assumptions* **A1'**, **A2** *and* **A3** *hold, then no complete master program solution* (η^i, y^i) *is generated twice by Algorithm 7.*

Proof:
(i) If NLP(y^i) is feasible then (η^i, y^i) must satisfy the constraints

$$\begin{aligned} \eta^i &\leq \text{UBD}^i - \epsilon_0^i && (8.1)\\ \eta^i &\geq c^i + (a^i)^T \begin{pmatrix} x - x^i \\ y^i - y^i \end{pmatrix} && (8.2)\\ 0 &\geq g^i + (g^i)^T \begin{pmatrix} x - x^i \\ y^i - y^i \end{pmatrix} && (8.3)\\ \text{UBD}^i &\leq f^i && (8.4) \end{aligned}$$

Summing (8.3) over all constraints weighted by their multipliers λ^i, adding to (8.2) and applying assumption **A1'** which ensures the existence of non-negative multipliers λ^i such that $a_x^i = [\nabla_x g^i]^T \lambda^i$ and ${\lambda^i}^T g^i = 0$ the cut

$$\eta^i + \epsilon_1^i + \epsilon_3^i \geq c^i \qquad (8.5)$$

is obtained (similarly to the finite convergence proof of Theorem 5.2.1). Inequality (8.1) implies that

$$\begin{aligned} c^i - \eta^i &\geq c^i - \text{UBD}^i + \epsilon_0^i \\ &\geq c^i - f^i + \epsilon_0^i \\ &\geq -\epsilon_2^i + \epsilon_0^i \\ &> \epsilon_1^i + \epsilon_3^i \end{aligned}$$

which contradicts (8.5).
(ii) If NLP(y^i) is infeasible then the proof follows in a similar fashion, using the techniques of Lemma 4.2.1.
□

The proof indicates why it is necessary to choose $\epsilon_0^i > \epsilon_1^i + \epsilon_2^i + \epsilon_3^i$. In order to ensure that no integer assignment is generated more than once, an additional assumption on ϵ_0^i is needed and this is stated in the following Theorem.

Theorem 8.3.1 *If assumptions* **A1'**, **A2** *and* **A3** *hold and if*

$$\epsilon_0^i \geq \max_{j \leq i} \epsilon_0^j$$

then Algorithm 7 converges finitely to an ϵ_0^i *optimal solution of P.*

Proof:
First it is shown that Algorithm 7 terminates finitely by proving that no integer assignment is generated twice. If $y = y^i$ gives rise to a feasible NLP subproblem, assume that the same integer assignment is generated again at iteration $l > i$. It follows that there exists a triple (η', x', y^i) which satisfies the constraints

$$\begin{aligned} \eta' &\leq \text{UBD}^l - \epsilon_0^l && (8.6) \\ c^i &\leq \eta' + \epsilon_1^i + \epsilon_3^i && (8.7) \\ \text{UBD}^l &\leq f^i, && (8.8) \end{aligned}$$

where (8.7) is obtained in the same way as (8.5) in the proof of Lemma 8.3.1. Forming $c^i - \eta'$ the following inequality is obtained

$$\begin{aligned} c^i - \eta' &\geq c^i - \text{UBD}^l + \epsilon_0^l \\ &\geq c^i - f^i + \epsilon_0^l \\ &\geq -\epsilon_2^i + \epsilon_0^l && (8.9) \end{aligned}$$

Combining (8.9) and (8.7) gives

$$-\epsilon_2^i + \epsilon_0^l \leq c^i - \eta' \leq \epsilon_1^i + \epsilon_3^i$$

which implies that $\epsilon_0^l \leq \epsilon_1^i + \epsilon_3^i + \epsilon_1^i$. This contradicts the choice of ϵ_0^l as the maximum over all ϵ_0^i which are strictly greater than the sum of the individual tolerances.
If $y = y^i$ gives rise to an infeasible subproblem, then the proof that y^i is not generated twice is similar to the above proof and uses the techniques developed in Chapter 5 for infeasible subproblems.
Since Y is a finite set and no integer assignment is generated twice, the algorithm terminates finitely. Moreover, since all linearizations are outer approximations of g (convex) and f by assumption **A1'**, each master program relaxation is a relaxation of P and the algorithm terminates at an ϵ_0^i optimal solution.
□

It is a consequence of Theorem 8.3.1 that Algorithm 7 is not only relevant to inexact MINLP, but may be used for solving nonconvex MINLP problems. Although it is not always possible to derive strict linear underestimators, Algorithms 7 is guaranteed to find the global optimum in a number of special cases such as convex MINLP and certain concave INLP problems in a finite number of steps, thereby improving on the two heuristics described in Section 8.2. The concept of linear underestimators also offers greater flexibility for heuristics in the sense that the user can decide how much effort to spend in deriving underestimators by providing a routine that finds linear underestimators.

8.4 Application to nonconvex MINLP problems

In this section the outer approximation framework developed in the previous section is used to solve certain nonconvex MINLP problems. First a deterministic method based on rigorous linear

underestimators is derived for the case where the objective function f is a concave function in the integer variables y. This motivates a heuristic for the general nonconvex case which is given towards the end of this section. Any algorithm that fits into the framework of Algorithm 7 is completely described by the derivation of the underestimator.

First a deterministic method for the following nonconvex MINLP model problem is proposed.

$$\begin{cases} \min\limits_{y} & f(y) \\ \text{subject to} & g(y) \leq 0 \\ & y \in Y \text{ integer.} \end{cases}$$

where f is concave. It is possible to derive valid linear underestimators of f in this case. Together with the usual outer approximation of the feasible region this gives rise to a finite algorithm.

Linear underestimators on f can be derived following a procedure proposed by Pardalos and Rosen [59]. Suppose there exists a set of points

$$R = \{y_1, \ldots, y_q\} \text{ where } q \geq p+1$$

such that the feasible set is contained in the convex hull of R, that is

$$\{y : g(y) \leq 0,\ y \in Y \text{ integer}\} \subseteq \text{conv}(R)$$

A linear underestimator of f over R is then also a linear underestimator of f over the entire feasible region and can be derived by solving a linear programming problem.

A linear programming problem that determines the linear underestimator

$$l(y) = a^T y + b$$

is formulated so that

$$l(y_i) \leq f^i := f(y_i)\ ,\ \forall i = 1, \ldots, q$$

and a "best" underestimator is obtained. This gives rise to the following LP

$$\begin{cases} \min\limits_{a,b} & \sum\limits_{i=1}^{q} (f^i - a^T y_i - b) \\ \text{subject to} & a^T y_i + b \leq f^i\ ,\ i = 1, \ldots, q. \end{cases}$$

Defining y' as the centroid of the polyhedron conv(R), that is

$$y' = \frac{1}{q} \sum_{i=1}^{q} y_i$$

and introducing the matrix $B = [y_1 : \ldots : y_q]$ and the vector $F = (f^1, \ldots, f^q)^T$, the LP can be written as

$$(U) \begin{cases} \max\limits_{a,b} & y'^T a + b \\ \text{subject to} & B^T a + be \leq F \end{cases}$$

where $e = (1, \ldots, 1)^T$. It is possible to show that equality holds in at least $p+1$ inequalities at the optimal solution to this LP by considering its dual, which is given by

$$\begin{cases} \min\limits_{\lambda} & \lambda^T F \\ \text{subject to} & y' = B\lambda \\ & 1 = e^T \lambda \\ & \lambda \geq 0. \end{cases}$$

The dual LP has $p+1$ equality constraints and hence its optimal solution has at least $p+1$ basic variables, so that at least $p+1$ multipliers λ are nonnegative and hence there are at least $p+1$ active constraints in the primal.

The set R can and in fact should be varied from iteration to iteration to reflect the addition of cuts to the master program relaxation. An initial choice for R is a set that contains Y. Tuy [71] reports on two procedures that update the vertex set of a polytope as new constraints are added to it. Alternatively, if there are no nonlinear constraints and $Y \subseteq \{0,1\}^p$, the set R can be taken to consist of exactly $p+1$ points which include the current iterate and choose p points lying along the n edges emanating from the current iterate. Instead of solving an LP only a linear non–singular system of equations needs to be solved in this case. It is clear that keeping R constant over several outer approximation iterations should be avoided, since a constant R results in the same linear underestimator being generated at several iterations.

The building blocks for this particular nonconvex MINLP approach fit into the outer approximation framework of Algorithm 7 by replacing step 2 by

2. Linearize the convex constraints $0 \geq g(y)$ about y^i and update the set of vertices R accordingly. Solve the LP to find a linear underestimator to f over $\text{conv}(R)$. Compute the approximation error $\epsilon_2^i = f^i - c^i$.

The tolerance ϵ_2^i is not necessarily a small number, but depends on how close the linear underestimator can be chosen to the objective function at y^i. The methods for deriving linear underestimators to concave functions are standard in the global optimization community, but seem not to have been applied to concave integer minimization problems. Outer approximation appears to provide a convenient framework for generalizing methods based on linear underestimators to integer and mixed integer problems.

While the methods above are rigorous for only a limited class of problems, it is certainly desirable to devise heuristic methods based on the outer approximation framework that are suitable for a more general class of problems. The class of problems considered here is the minimization of a nonconvex function of mixed integer variables over a convex set, that is

$$\begin{cases} \min\limits_{x,y} & f(x,y) \\ \text{subject to} & g(x,y) \leq 0 \\ & x \in X \ , \ y \in Y \text{ integer.} \end{cases}$$

where f is a nonconvex function and g are convex constraint functions.

The central idea is to approximate the nonconvex objective function by a quadratic function about the current iterate (x^i, y^i) and then to derive a linear underestimator for this quadratic. The limitations of this process are clear: the quadratic model might not be an underestimator of f and the NLP(y^i) subproblems cannot be expected to have been solved to optimality. However, the advantage over the other outer approximation heuristics is that a quadratic underestimator offers a greater flexibility and that Algorithm 7 has a finite termination property.

In the mixed integer case the assumption **A1'** requires the existence of multipliers u^i such that the objective gradient is a multiple of a convex combination of the constraint gradients. It is convenient to include this condition in the LP problem that is solved to obtain the linear

underestimator. Hence the following LP is solved instead of U

$$\begin{cases} \max\limits_{a,b,u} & (x',y')^T a + b \\ \text{subject to} & B^T a + be \leq F \\ & a_x = -\sum\limits_{j \in \mathcal{A}^i} u_j \nabla_x g_j^i \\ & u_j \geq 0 \ , \ \forall j \in \mathcal{A}^i \\ & u_j = 0 \ , \ \forall j \notin \mathcal{A}^i \end{cases}$$

where $\mathcal{A}^i$ is the active set in iteration i of the corresponding nonlinear program. A simpler choice for a_x is to take some positive multiple of $\nabla_x f^i$. This choice always gives rise to a linear underestimator for sufficiently large b. However, including the computation of u and a_x in the LP allows a greater flexibility in the derivation of linear underestimators and should therefore result in a tighter underestimator.

It is noted in Section 8.3 that Algorithm 7 might generate the same integer assignment again if the overall accuracy ϵ_0^i decreases. In the context of nonconvex MINLP problems this might be desirable, since it offers the possibility of restarting the nonconvex NLP(y^i) from a new initial x. However, in order to ensure finiteness of Algorithm 7 it is necessary to enforce monotonicity of ϵ_0^i so that no integer assignment is generated more than once.

8.5 Application to inexact NLP/MILP/MIQP solvers

The outer approximation algorithm of Duran and Grossmann [15] makes the implicit assumption that exact NLP and MILP solvers are available and the finite termination proof of Chapter 5 makes a similar assumption. In practice, this assumption is not true since NLP, MILP and MIQP solvers work to a given tolerance or precision. In this chapter the influence of these inaccuracies upon the finite termination property of an outer approximation scheme is studied. In particular, the tolerances ϵ_1^i and ϵ_3^i need to reflect the accuracy of the NLP and MILP solver respectively in order to guarantee finite termination. Lower bounds on ϵ_1^i and ϵ_3^i are derived based solely on the tolerances of the respective solvers. For the sake of simplicity it is assumed, that f is again convex.

Before considering the case of general integer variables it is interesting to consider the mixed binary problem. If all integer variables are binary then at each iteration of the outer approximation algorithm an integer cut is added to the master program. For $y = y^i$ the integer cut that excludes y^i from subsequent master programs is

$$\rho^T y \leq b$$

where

$$\rho_j = \begin{cases} 1 & \text{if } y_j^i = 1 \\ -1 & \text{if } y_j^i = 0 \end{cases}$$

$$b = \sum_{j=1}^{p} y_j^i - 1.$$

The integer cut not only excludes the assignment y^i but also any arbitrary perturbation of y^i and the right hand side b given by

$$y_j' = \begin{cases} 1-\epsilon & \text{if } y_j^i = 1 \\ \epsilon & \text{if } y_j^i = 0 \end{cases}$$

provided that ϵ is not too large. If ϵ is the tolerance to which the MILP or MIQP solver works, then in order to exclude *any* perturbation y' of y^i it is necessary that

$$\epsilon < \frac{1}{p+1}$$

so that for a single precision MILP solver with $\epsilon = 10^{-6}$, perturbations of the above kind are tolerated for up to $p < 10^6 - 1$ variables. This implies that an integer cut excludes any perturbation of y^i from the master program as long as the number of binary variables does not exceed the reciprocal of the tolerance of the MILP or MIQP solver. Even for a single precision MILP or MIQP solver the number of binary variables that can safely be handled by far exceeds the size of problem that one can expect to solve in a reasonable amount of time.

This shows that for the near future integer cuts are sufficient to exclude binary assignments from being generated again by the outer approximation algorithm. However, if integer cuts are not available then an integer assignment may be generated again if ϵ_0^i is not large enough. For example a rather silly choice for ϵ_0^i would be to set it to a smaller value than the tolerance of the MILP solver which implies that in practice *no* integer assignment is ever excluded from the master program. The remainder of this section considers the role of tolerances for MILP and NLP solvers separately.

MILP and MIQP solvers work to a finite precision ϵ and the constraints are only satisfied approximately to within this tolerance. In order to derive lower bounds on ϵ_3^i a perturbation to an integer assignment y^i is considered. If the perturbation y' is feasible to within the tolerance ϵ then the following constraints are satisfied

$$\begin{aligned} \|y' - y^i\| &\leq \epsilon \\ \eta + \epsilon &\geq f^i + (\nabla f^i)^T \begin{pmatrix} x - x^i \\ y' - y^i \end{pmatrix} \qquad (8.10) \\ e\epsilon &\geq g^i + [\nabla g^i]^T \begin{pmatrix} x - x^i \\ y' - y^i \end{pmatrix} \qquad (8.11) \end{aligned}$$

summing the constraints (8.11) weighted with their respective multipliers and adding to (8.10) the following cut is obtained (after applying Kuhn–Tucker conditions to the x part)

$$\eta + \epsilon(1 + e^T\lambda^i) \geq f^i + (\nabla_y f^i + \nabla_y g^i \lambda^i)^T (y' - y^i).$$

If ϵ_2^i is chosen such that

$$\epsilon_2^i \geq \epsilon(1 + e^T\lambda^i + \|\nabla_y f^i + \nabla_y g^i \lambda^i\|) \qquad (8.12)$$

then at worst

$$\eta + \epsilon_2^i \geq f^i \;\Rightarrow\; \eta \geq f^i - \epsilon_2^i.$$

Since $\epsilon_o^i > \epsilon_2^i$, the upper bound constraint in the master program

$$\eta \leq \text{UBD} - \epsilon_0^i \leq f^i - \epsilon_0^i$$

ensures that even the perturbation y' of the integer assignment y^i is not feasible in any subsequent master program. Expression (8.12) gives a lower bound on the tolerance ϵ_2^i and shows that it is a multiple of the tolerance ϵ of the MILP or MIQP solver, where ϵ depends on the size of the multipliers and the norm of the gradients with respect to the y variables. Both quantities can be computed at little additional cost upon termination of the NLP solver.

The problem of inaccurate solutions is not only restricted to the MILP or MIQP solver but also concerns the solution of the NLP subproblems. Usually, the NLP solver does not solve the NLP subproblem exactly, but obtains a solution which is an approximate Kuhn–Tucker point. That is for a given tolerance ϵ, the NLP solver finds a solution x^i to NLP(y^i) that satisfies

$$\begin{array}{ll} \epsilon \geq f^i - f^{i^*} & \text{function error; } f^{i^*} \text{ optimum} \\ \epsilon \geq \|\nabla_x f^i + \nabla_x g^i \lambda^i\| & \text{Kuhn–Tucker error} \\ \epsilon \geq \max\limits_{0 \leq j \leq m} (0, g_j^i) & \text{infeasibility error.} \end{array}$$

Consequently, in order to ensure that y^i is not feasible in any subsequent master program relaxations, ϵ_1^i has to be large enough not to be upset by the errors in the NLP solution. A lower bound on ϵ_1^i can again be derived by considering the relevant linearizations in the master program, that is

$$\begin{aligned} \eta &\leq \text{UBD} - \epsilon_0^i \\ \eta &\geq f^i + (\nabla f^i)^T \begin{pmatrix} x - x^i \\ y^i - y^i \end{pmatrix} \qquad (8.13) \\ 0 &\geq g^i + [\nabla g^i]^T \begin{pmatrix} x - x^i \\ y^i - y^i \end{pmatrix} \qquad (8.14) \end{aligned}$$

Summing the constraint (8.13) weighted with their respective multipliers and adding to (8.14) the following valid inequality is obtained

$$\eta \geq f^i + {\lambda^i}^T g^i + (\nabla_x f^i + \nabla_x g^i \lambda^i)^T (x - x^i).$$

Since the NLP(y^i) subproblem has not been solved exactly, neither ${\lambda^i}^T g^i$ nor $\nabla_x f^i + \nabla_x g^i \lambda^i$ are zero and in theory this could result in the same integer assignment being picked again with a different x value. If, however, ϵ_1^i is chosen such that

$$\epsilon_1^i \geq \epsilon (e^T \lambda^i + \max_{x \in X} \|x - x^i\|) \qquad (8.15)$$

then at worst $\eta \geq f^i - \epsilon_1^i$ which contradicts the upper bound constraint $\eta \leq f^i - \epsilon_0^i$ since $\epsilon_0^i > \epsilon_1^i$. The lower bound on ϵ_1^i given in (8.15) is a multiple of the tolerance of the NLP solver and can be computed after the NLP subproblem has been solved. The dependence of the lower bound on the $\max \|x - x^i\|$ term indicates that it is advisable to solve the NLP subproblems to a high accuracy in order to ensure a small ϵ_0^i.

The remarks in this section indicate that the overall accuracy of the outer approximation algorithm should *not* be set arbitrarily by the user but depends on the accuracy of the NLP and MILP/MIQP solvers and on the tightness of the linearizations used. A rigorous implementation should take this into account and modify the overall accuracy ϵ_0^i according to the rules presented above. Upon termination, the final ϵ_0^i together with the solution should be provided. Failure to take the inaccuracies into account could result in an implementation that might cycle between integer assignments.

Chapter 9

Conclusions

This thesis is concerned with deterministic algorithms for the solution of smooth and convex Mixed Integer Nonlinear Programming problems (MINLPs) and a number of algorithms have been studied in detail, as well as extensions to nonsmooth and nonconvex MINLP problems.

One such algorithm is nonlinear branch–and–bound of which various aspects have been studied. In particular a branch–and–bound algorithm for solving MIQP problems has been improved by new improved lower bounds for child nodes. The lower bounds are computed by taking a step of the dual active set method parametrically. The numerical experience that has been presented suggests that the new lower bounds are a useful addition to an MIQP branch–and–bound solver.

One main theme of this thesis has been to study the outer approximation algorithms of Duran and Grossmann [15] in detail. This is a cutting plane algorithm which solves a convex and smooth MINLP by iterating finitely between an MILP master program and an NLP subprogram. It thus separates the combinatorial part from the nonlinear part of the problem. The derivation of the MILP master program has been simplified and inaccuracies concerning the treatment of infeasible subproblems have been corrected. The new derivation affords additional insight into outer approximation and this has been exploited throughout the thesis. The rigorous treatment of infeasible subproblems allows a correct master program to be given, and five algorithms have been derived based upon this master program.

A worst case example has been presented for which outer approximation visits *all* feasible integer assignments in turn before finding the solution. This behaviour has been explained by the failure of outer approximation to take curvature information into account and has motivated the introduction of second order term into the master program, resulting in a quadratic outer approximation algorithm. This has motivated the development of an MIQP solver mentioned above.

Another main idea has been to study the LP/NLP based branch-and–bound algorithm of Quesada and Grossmann [60]. This algorithm is valuable in that it avoids the re–solution of successive related master program relaxations. It has been shown that this algorithm can be extended to MINLP problems in which the integer variables occur nonlinearly. The new algorithm exhibits the same worst case behaviour as outer approximation and a new algorithm which includes second order information in the master programs has been suggested. Finite convergence results are shown for all algorithms.

Two related algorithms have also been studied. The first, Generalized Benders Decomposition, has been compared to outer approximation and has been shown to generate weaker cuts than outer approximation. The new derivation of the MILP master program also offers a means of generalizing Lagrangian Decomposition to MINLPs with nonlinear constraints and a new Lagrangian Decomposition algorithm has been proposed and finite convergence properties have been proved.

It has been argued that the new scheme is unlikely to perform better than outer approximation in practice, except for very special problems.

The practical performance of the four algorithms based upon outer approximation (linear and quadratic outer approximation and LP and QP/NLP based branch–and–bound) has been compared to an implementation of a nonlinear branch–and–bound algorithm and the main conclusion has been that the performance is problem dependent. The test problems can be grouped into three broad classes and a rule of thumb for choosing the best MINLP solver for any MINLP is motivated from the results of the experiment. LP/NLP based branch–and–bound is seen to perform best for binary problems, Nonlinear Branch–and–Bound works best on general integer problems with a small integrality gap and QP/NLP based branch–and–bound outperforms the other routines for general integer problems with a large integrality gap.

The outer approximation scheme has been generalized to cover a certain class of nonsmooth MINLP problems by applying the theory of subgradients to this particular situation. A special class of nonsmooth MINLP problems which arise through an exact penalty formulation of a smooth MINLP problem has been studied in detail. Finite convergence results have been given as well as lower bounds on the size of the penalty parameter.

Outer approximation requires a convexity assumption to be made which can be very restrictive in practice. By introducing the use of tolerances into the outer approximation algorithm a new more flexible outer approximation framework has been derived and finite convergence to an ϵ–optimal solution has been proved. The new framework can be applied to a certain class of nonconvex MINLP problems. If the objective function is a pure integer concave function which is minimized over a convex set then a finitely ϵ–optimal convergent algorithm has been presented which makes use of rigorous linear underestimators. This idea has been extended to give a finitely terminating heuristic algorithm for a class of nonconvex MINLP problems and the convex MINLP case is recovered as a special case.

The new framework also enables the finite termination result to be extended to the case where inexact NLP, QP, LP, MIQP or MILP solvers are used. Lower bounds on the tolerance ϵ have been given that ensure finite termination.

The new flexibility introduced by the outer approximation framework affords in my view the best possibility of deriving globally convergent MINLP algorithms for classes of nonconvex problems. It also offers the most promising framework for deriving good heuristic methods for classes of nonconvex MINLP problems for which there is no deterministic approach.

Appendix A

MINLP Test Problems

This appendix lists descriptions of the MINLP test problems which have been used in the numerical experiments of Chapter 6. The description includes the integer solution (in the line headed "integer solution") and the solution of the NLP relaxation (in the line headed "NLP solution") for each problem.

Test problems TP1 through to TP3 are taken from [15]. They constitute small process synthesis examples and all fall into the same class of MINLP problems that are linear in y.

Test problem TP1

$$
\left\{
\begin{array}{ll}
\min\limits_{x,y} & 5y_1 + 6y_2 + 8y_3 + 10x_1 - 7x_3 - 18\ln(x_2+1) \\
 & -19.2\ln(x_1 - x_2 + 1) + 10 \\
\text{subject to} & 0.8\ln(x_2+1) + 0.96\ln(x_1 - x_2 + 1) - 0.8x_3 \quad \geq 0 \\
 & \ln(x_2+1) + 1.2\ln(x_1 - x_2 + 1) - x_3 - 2y_3 \quad \geq -2 \\
 & x_2 - x_1 \quad \leq 0 \\
 & x_2 - 2y_1 \quad \leq 0 \\
 & x_1 - x_2 - 2y_2 \quad \leq 0 \\
 & y_1 + y_2 \quad \leq 1 \\
 & 0 \leq x \leq u, \quad \text{where } u^T = (2,2,1) \\
 & y \in \{0,1\}^3 \\
\text{NLP solution} & f' = 0.759,\ y' = (0.273, 0.300, 0.000)^T \\
 & x' = (1.147, 0.547, 1.000)^T \\
\text{integer solution} & f^* = 6.010,\ y^* = (1,0,1)^T \\
 & x^* = (1.301, 0.000, 1.000)^T
\end{array}
\right.
$$

Test problem TP2

$$
\left\{
\begin{array}{ll}
\min\limits_{x,y} & 5y_1 + 8y_2 + 6y_3 + 10y_4 + 6y_5 - 10x_1 - 15x_2 - 15x_3 + 15x_4 \\
& +5x_5 - 20x_6 \\
& + \exp(x_1) + \exp(x_2/1.2) - 60\ln(x_4 + x_5 + 1) + 140 \\
\text{subject to} & -\ln(x_4 + x_5 + 1) \leq 0 \\
& \exp(x_1) - 10y_1 \leq 1 \\
& \exp(x_2/1.2) - 10y_2 \leq 1 \\
& 1.25x_3 - 10y_3 \leq 0 \\
& x_4 + x_5 - 10y_4 \leq 0 \\
& -2x_3 + 2x_6 - 10y_5 \leq 0 \\
& -x_1 - x_2 - 2x_3 + x_4 + 2x_6 \leq 0 \\
& -x_1 - x_2 - 0.75x_3 + x_4 + 2x_6 \leq 0 \\
& x_3 - x_6 \leq 0 \\
& 2x_3 - x_4 - 2x_6 \leq 0 \\
& -0.5x_4 + x_5 \leq 0 \\
& 0.2x_4 - x_5 \leq 0 \\
& y_1 + y_2 = 1 \\
& y_4 + y_5 \leq 1 \\
& 0 \leq x \leq u, \quad \text{where } u^T = (2, 2, 2, \infty, \infty, 3) \\
& y \in \{0, 1\}^5 \\
\text{NLP solution} & f' = -0.554 \\
& y' = (0.571, 0.429, 0.250, 0.210, 0.)^T \\
& x' = (1.903, 2.000, 2.000, 1.403, 0.701, 2.000)^T \\
\text{integer solution} & f^* = 73.035 \\
& y^* = (0, 1, 1, 1, 0)^T \\
& x^* = (0.000, 2.000, 1.078, 0.652, 0.326, 1.078)^T
\end{array}
\right.
$$

Test problem TP3
Objective function:

$$\begin{aligned} f(x,y) = \quad & 5y_1 + 8y_2 + 6y_3 + 10y_4 + 6y_5 + 7y_6 + 4y_7 + 5y_9 \\ & -10x_1 - 15x_2 + 15x_3 + 80x_4 + 25x_5 + 35x_6 - 40x_7 \\ & +15x_8 - 35x_9 + \exp(x_1) + \exp(x_2/1.2) - 65\ln(x_3 + x_4 + 1) \\ & -90\ln(x_5 + 1) - 80\ln(x_6 + 1) + 120 \end{aligned}$$

Constraints:

$$\left\{ \begin{array}{rl} -1.5\ln(x_5+1) - \ln(x_6+1) - x_8 & \leq 0 \\ -\ln(x_3+x_4+1) & \leq 0 \\ -x_1 - x_2 + x_3 + 2x_4 + 0.8x_5 + 0.8x_6 - 0.5x_7 - x_8 - 2x_9 & \leq 0 \\ -x_1 - x_2 + 2x_4 + 0.8x_5 + 0.8x_6 - 2x_7 - x_8 - 2x_9 & \leq 0 \\ -2x_4 - 0.8x_5 - 0.8x_6 + 2x_7 + x_8 + 2x_9 & \leq 0 \\ -0.8x_5 - 0.8x_6 + x_8 & \leq 0 \\ -x_4 + x_7 + x_9 & \leq 0 \\ -0.4x_5 - 0.4x_6 + 1.5x_8 & \leq 0 \\ 0.16x_5 + 0.16x_6 - 1.2x_8 & \leq 0 \\ x_3 - 0.8x_4 & \leq 0 \\ -x_3 + 0.4x_4 & \leq 0 \\ \exp(x_1) - 10y_1 & \leq 1 \\ \exp(x_2/1.2) - 10y_2 & \leq 1 \\ x_7 - 10y_3 & \leq 0 \\ 0.8x_5 + 0.8x_6 - 10y_4 & \leq 0 \\ 2x_4 - 2x_7 - 2x_9 - 10y_5 & \leq 0 \\ x_5 - 10y_6 & \leq 0 \\ x_6 - 10y_7 & \leq 0 \\ x_3 + x_4 - 10y_8 & \leq 0 \\ y_1 + y_2 & = 1 \\ y_4 + y_5 & \leq 1 \\ -y_4 + y_6 + y_7 & = 0 \\ y_3 - y_8 & \leq 0 \\ 0 \leq x \leq u, \quad \text{where } u^T = (2,2,1,2,2,2,2,1,3) & \\ y \in \{0,1\}^8 & \end{array} \right.$$

NLP solution

$$\begin{aligned} f' &= 15.082 \\ y' &= (0.571, 0.430, 0.066, 0.308, 0.000, 0.200, 0.108, 0.119)^T \\ x' &= (1.903, 2.000, 0.528, 0.659, 2.000, 1.083, 0.659, 0.411, 0.000)^T \end{aligned}$$

Integer solution

$$\begin{aligned} f^* &= 68.010 \\ y^* &= (0, 1, 0, 1, 0, 1, 0, 1)^T \\ x^* &= (0.000, 2.000, 0.468, 0.585, 2.000, 0.000, 0.000, 0.267, 0.585)^T \end{aligned}$$

The next two test problems are reformulations of an optimal design problem for a multiproduct batch plant. In the original problem the integer variables model the number of parallel units. Kocis and Grossmann [42] replace these by binary variables and remove the nonconvexities by applying

a logarithmic transformation to the variables and this gives rise to BATCH. An alternative formulation which is nonconvex and hence does not give the same guarantees as Kocis and Grossmann's is also presented (BATCH-int). In this formulation the integer variables are not replaced by binary variables, and hence it is not possible to convexify the original objective function which contains nonconvex terms of the form $N_j V_J^{\beta_j}$.

Both problems minimize the investment cost subject to volume, cycle and horizon constraints. The following table gives a definition of the variables. The number of stages is $M = 6$ and the number of products is $N = 5$. The parameters S_{ij}, t_{ij}, Q_i and H are given. The index j is used throughout for stages, the index i is used to index the products.

variable	description
n_j	Logarithm of the number of parallel units of stage j
v_j	Logarithm of the volume of stage j
b_i	Logarithm of the batch size of batch i
t_{L_i}	Logarithm of the cycle time of batch i
N_j	Number of parallel units at stage j
Y_{kj}	Binary variables used in BATCH to represent the integers N_j,

Test problem BATCH

$$
\begin{cases}
\min\limits_{n,v,t_L,b,y} & \sum\limits_{j=1}^{M} \alpha_j \exp(n_j + \beta_j v_j) \\
\text{subject to} & v_j \geq \ln(S_{ij}) + b_i,\ \forall i = 1,\ldots,N,\ \forall j = 1,\ldots,M \\
& n_j + t_{L_i} \geq \ln(t_{ij}),\ \forall i = 1,\ldots,N,\ \forall j = 1,\ldots,M \\
& \sum\limits_{i=1}^{N} Q_i \exp(t_{L_i} - b_i) \leq H \\
& \sum\limits_{k=1}^{N_j^U} \ln(k) y_{kj} = n_j,\ \forall j = 1,\ldots,M \\
& \sum\limits_{k=1}^{N_j^U} y_{kj} = 1,\ \forall j = 1,\ldots,M \\
& 0 \leq n_j \leq \ln(N_j^U),\ \forall j = 1,\ldots,M \\
& ln(V_j^L) \leq v_j \leq \ln(V_j^U),\ \forall j = 1,\ldots,M \\
& ln(T_{L_j}^L) \leq t_{l_i} \leq \ln(T_{L_i}^U),\ \forall i = 1,\ldots,N \\
& ln(B_i^L) \leq b_i \leq \ln(B_i^U),\ \forall i = 1,\ldots,N \\
& y_{kj} \in \{0,1\},\ \forall j = 1,\ldots,M,\ \forall k = 1,\ldots,N_j^U \\
\text{NLP solution} & f' = 259181 \\
& y' = (0.410, 0.356, 0.093, 0.141, 0.766, 0.041, 0.043, 0.150, \\
& 0.348, 0.048, 0.231, 0.373, 0.475, 0.059, 0.142, 0.323, \\
& 0.995, 0.000, 0.003, 0.002, 0.869, 0.014, 0.043, 0.073)^T \\
\text{integer solution} & f^* = 285506 \\
& y^* = (0100, 0100, 0010, 0100, 1000, 1000)^T
\end{cases}
$$

It is possible to avoid the introduction of the $M(\sum N_j^U)$ integer variables y_{kj} resulting in a much smaller problem with a *nonconvex* objective function. Nevertheless, all five codes found an optimal solution to the simpler formulation.

Test problem BATCH–int

$$
\begin{cases}
\min\limits_{N,v,t_L,b} & \sum\limits_{j=1}^{M} \alpha_j \exp(\ln(N_j) + \beta_j v_j) \\
\text{subject to} & v_j \geq \ln(S_{ij}) + b_i,\ \forall i = 1,\ldots,N,\ \forall j = 1,\ldots,M \\
& \ln(N_j) + t_{L_i} \geq \ln(t_{ij}),\ \forall i = 1,\ldots,N,\ \forall j = 1,\ldots,M \\
& \sum\limits_{i=1}^{N} Q_i \exp(t_{L_i} - b_i) \leq H \\
& 1 \leq N_j \leq N_j^U,\ \text{ integer }\ \forall j = 1,\ldots,M \\
& ln(V_j^L) \leq v_j \leq \ln(V_j^U),\ \forall j = 1,\ldots,M \\
& ln(T_{L_i}^L) \leq t_{l_i} \leq \ln(T_{L_i}^U),\ \forall i = 1,\ldots,N \\
& ln(B_i^L) \leq b_i \leq \ln(B_i^U),\ \forall i = 1,\ldots,N \\
\text{NLP solution} & f' = 259181, N' = (1.724, 1.327, 2.235, 1.908, 1.006, 1.172)^T \\
\text{integer solution} & f^* = 285506, N^* = (2, 2, 3, 2, 1, 1)^T
\end{cases}
$$

The next three test problems are taken from a paper by Asaadi [2]. The first problem is due to Rosen and Suzuki [61] and the latter two problems are due to Wong [77]. All three are nonlinear programming test problems which can easily be made into MINLP test problems by adding integer restrictions to some of the variables. A detailed description of these problems is given in the following.

Test Problem Asaadi 1

$$
\begin{cases}
\min\limits_{x} & x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4 \\
\text{subject to} & -x_1^2 - x_2^2 - x_3^2 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8 \geq 0 \\
& -x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 + 10 \geq 0 \\
& -2x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5 \geq 0 \\
\text{NLP solution} & f' = -40.963, x' = (0.000, 1.038, 2.227, 0.000)^T \\
\text{integer solution} & \text{Asaadi 1 (3) } f^* = -40.957, x^* = (0, 1, 2.236, 0)^T \\
\text{integer solution} & \text{Asaadi 1 (4) } f^* = -38.000, x^* = (0, 1, 2, 0)^T
\end{cases}
$$

Test Problem Asaadi 2

$$
\begin{cases}
\min\limits_{x} & (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 \\
& +10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7 \\
\text{subject to} & -2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 + 127 \geq 0 \\
& 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 + 282 \geq 0 \\
& 23x_1 - x_2^2 - 6x_6^2 + 8x_7 + 196 \geq 0 \\
& -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0 \\
\text{NLP solution} & f' = 683.981 \\
& x' = (2.348, 1.935, 0.000, 4.298, 0.000, 1.048, 1.582)^T \\
\text{integer solution} & \text{Asaadi 2 (4) } f^* = 694.90 \\
& x^* = (2, 2, 0, 4, -4.64 \times 10^{-4}, 1.132, 1.463)^T \\
\text{integer solution} & \text{Asaadi 2 (7) } f^* = 700.0 \\
& x^* = (2, 2, 0, 4, 0, 1, 2)^T
\end{cases}
$$

Test Problem Asaadi 3

$$\begin{cases} \min\limits_{x} & x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 \\ & +4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 \\ & +7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45 \\ \text{subject to} & -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0 \\ & -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0 \\ & -\frac{1}{2}(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0 \\ & -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0 \\ & 3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0 \\ & -4x_1 - 5x_2 + 3x_7 - 9x_8 + 105 \geq 0 \\ & -10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0 \\ & 8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0 \\ \text{NLP solution} & f' = 24.306 \\ & x' = (2.172, 2.364, 8.774, 5.096, 0.991, 1.431, 1.322, 9.829, \\ & \quad 8.280, 8.376)^T \\ \text{integer solution} & \text{Asaadi 3 (6) } f^* = 37.219 \\ & x^* = (2, 2.600, 8, 5.000, 1, 1.387, 2, 10, 8, 8.600)^T \\ \text{integer solution} & \text{Asaadi 3 (10) } f^* = 43.0 \\ & x^* = (2, 2, 8, 5, 1, 2, 2, 10, 8, 8)^T \end{cases}$$

Cha and Mayne [11] propose a small two dimensional MINLP example which is described next.

Test Problem 2DEx

$$\begin{cases} \min\limits_{x} & 2x_1^2 + x_2^2 - 16x_1 - 10x_2 \\ \text{subject to} & x_1^2 - 6x_1 + x_2 - 11 \leq 0 \\ & -x_1x_2 + 3x_2^2 + \exp(x_1 - 3) - 1 \leq 0 \\ & 0 \leq x_1 \leq 5 \\ & 3 \leq x_2 \leq 5 \\ & X_1, x_2 \text{ discrete with discreteness } 0.25 \\ \text{NLP solution} & f' = -56.944, x' = (3.947, 4.776)^T \\ \text{discrete solution} & f^* = -56.938, x^* = (4.00, 4.75)^T \end{cases}$$

Sandgren [65] solves a small gear train design problem. The aim is to produce a gear ratio which is as close as possible to a given ratio (1/6.931). The number of teeth of each gear is required to lie between 12 and 60. Sandgren proposes to minimize the square of the difference between the desired and the designed ratio which is a nonconvex MINLP problem.

Test Problem GTD

$$\begin{cases} \min\limits_{x} & (\frac{1}{6.931} - \frac{x_3x_2}{x_1x_4}) \\ \text{subject to} & 12 \leq x_i \leq 60, \text{ integer} \\ \text{NLP solution} & f' = 9.855 \times 10^{-20}, x' = (31.592, 12.000, 12.000, 31.592)^T \\ \text{integer solution} & f^* = 7.779 \times 10^{-5}, x^* = (32, 12, 12, 31)^T \end{cases}$$

The two problems **GTD chain(2)** and **GTD chain(3)** are chained versions of the above problem obtained by forming

$$f(x) = f(x_1, \ldots, x_4) + \sum_{i=2}^{p} f(x_{3i+1}, \ldots, x_{3i+4}),$$

where $p = 2, 3$ respectively. The indices of the integer variables and the solutions are as follows.

Test Problem GTD chain(2) and (3)

	GTD chain(2)	GTD chain(3)
integer indices	$1, 2, 4, 5, 6$	$1, 2, 4, 5, 6, 7, 8, 9$
NLP solution	$f' = 4.105 \times 10^{-19}$ $x' = (40.378, 16.966, 16.966,$ $49.410, 16.966, 16.966, 40.378)^T$	$f' = 8.216 \times 10^{-16}$ $x' = (40.488, 16.863, 16.863,$ $48.680, 18.491, 18.491, 48.680,$ $16.863, 16.863, 40.488)^T$
integer solution	$f^* = 2.812 \times 10^{-23}$ $x^* = (40, 17, 16.634, 49,$ $17, 17, 40.879)^T$	$f^* = 4.9148 \times 10^{-6}$ $x^* = (40, 17, 16.634, 49,$ $18, 20, 51, 17, 18, 41.58597)^T$

The remaining MINLP test problems are MIQP problems. They were generated by taking two LP problems, adding a positive definite Hessian to the objective function and requiring some variables to take integer values. The first MIQP problem is a modification of AVGAS and the second is a variation of AFIRO, which is taken from the SOL test set. The constraints for AVGAS are given below as well as the Hessian used with AFIRO and their respective solutions.

Test Problem AVGAS 1

$$\begin{cases} \min\limits_{x} & -2x_1 - x_2 - 2x_3 - 3x_4 - 4x_5 - 5x_6 - 6x_7 + 8x_8 + \frac{1}{2}x^T G x \\ \text{subject to} & -x_1 - x_2 \geq -1 \\ & -x_3 - x_4 \geq -1 \\ & -x_5 - x_6 \geq -1 \\ & -x_7 - x_8 \geq -1 \\ & -x_1 - x_3 - x_5 - x_7 \geq -2 \\ & -x_2 - x_4 - x_6 - x_8 \geq -2 \\ & 2x_1 + x_3 - x_7 \geq 0 \\ & 5x_1 + 3x_3 - 3x_5 - x_7 \geq 0 \\ & x_2 - x_4 - 3x_6 - 5x_8 \geq 0 \\ & x_2 - 3x_6 - 2x_8 \geq 0 \\ \text{NLP solution} & f' = -8.114 \\ & x' = (0.241, 0.759, 0.389, 0.473, 0.500, 0.095, 0.870, 0.000)^T \\ \text{integer solution} & f^* = -4.000 \\ & x^* = (1, 0, 0, 0, 0, 0, 1, 0)^T \end{cases}$$

where $G = tri(-1, 4, -1)$.

Test Problem AVGAS 2

$$\begin{cases} \text{NLP solution} & f' = -6.631 \\ & x' = (0.403, 0.398, 0.146, 0.303, 0.500, 0.0320.951, 0.000)^T \\ \text{integer solution} & f^* = -4.000 \\ & x^* = (0, 0, 1, 0, 0, 0, 1, 0)^T \end{cases}$$

where $G = tri(1, 4, 1)$.

The Hessian matrix for the last suite of test problems is a tri–diagonal matrix formed from the diagonal vector

$$a = (4, 4, 4, 2, 4, 2, 4, 4, 4, 2, 4, 4, 4, 4, 2, 4, 2, 4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 2, 4, 4, 4, 2, 4)^T$$

and its off–diagonal entries are all -1. The next table gives the integer indices for the different AFIRO problems.

Test Problem AFIRO (3), (5), (6), (7)

AFIRO	integer indices
(3)	$1, 3, 5$
(5)	$1, 3, 5, 15, 18$
(6)	$1, 3, 5, 15, 18, 22$
(7)	$1, 3, 5, 15, 18, 22, 41$

The following two tables give the objective value of the NLP solution, f', and the objective value of the MINLP solution, f^*, and the corresponding integer vectors, respectively.

AFIRO	f'	f^*
(3)	925.90	932.37
(5)	925.90	988.92
(6)	925.90	989.26
(7)	972.28	1030.64

AFIRO	x'	x^*
(3)	$(0.435, 0.435, 1.675, 1.776)^T$	$(1, 1, 2)^T$
(5)	$(0.435, 0.435, 1.675, 1.776, 0.131)^T$	$(1, 1, 1, 2, 1)^T$
(6)	$(0.435, 0.435, 1.675, 1.776, 0.131, 1.603)^T$	$(1, 1, 1, 2, 1, 1)^T$
(7)	$(0.335, 0.335, 1.268, 1.344, 0.101, 1.119, 5.000)^T$	$(1, 1, 1, 2, 1, 0, 5)^T$

Bibliography

[1] H.M. Amir and T. Hasegawa, "Nonlinear Mixed-Discrete Structural Optimization", *Journal of Structural Engineering* **115** (1989) 626–646.

[2] J. Asaadi, "A computational comparison of some non–linear programs", *Mathematical Programming* **4** (1973) 144–154.

[3] E. Balas and J.B. Mazzola, "Nonlinear $0-1$ programming: I. Linearization techniques", *Mathematical Programming* **30** (1984) 1-21.

[4] E. Balas and J.B. Mazzola, "Nonlinear $0-1$ programming: II. Dominance relations and algorithms", *Mathematical Programming* **30** (1984) 22–45.

[5] E.M.L. Beale, "Integer Programming", in *The State of the Art in Numerical Analysis* (Editor D.A.H. Jacobs) (Academic Press, London, 1978).

[6] J.F. Benders, "Partitioning procedures for solving mixed–variable programming problems", *Numerische Mathematik* **4** (1962) 238–252.

[7] M. Benichou, J.M. Gauthier, P. Girodet, G. Hentges, G. Ribiere and O. Vincent, "Experiments in Mixed–Integer Linear Programming", *Mathematical Programming* **1** (1971) 76-94.

[8] M. Bremicker, P.Y. Papalambros and H.T. Loh, "Solution of mixed–discrete structural optimization problems with a new sequential linearization algorithm", *Computers and Structures* **37** (1990) 451–461.

[9] R. Breu and C.-A. Burdet, "Branch–and–bound experiments in 0–1 programming", *Mathematical Programming Study* **2** (1974) 1–50.

[10] J.Z. Cha and R.W. Mayne, "Optimization with discrete variables via recursive quadratic programming: Part 1 – concepts and definitions", *Transactions of the ASME, Journal of Mechanisms, Transmissions, and Automation in Design* **111** (1989) 124–129.

[11] J.Z. Cha and R.W. Mayne, "Optimization with discrete variables via recursive quadratic programming: Part 2 – algorithms and results", *Transactions of the ASME, Journal of Mechanisms, Transmissions, and Automation in Design* **111** (1989) 130–136.

[12] S. Chandra and L.C.W. Dixon, "Benders' decomposition for the constraint l_1-problem", *Journal of Optimization Theory and Applications* **68** (1991) 217–232.

[13] R.J. Dakin, "A tree search algorithm for mixed integer programming problems", *Computer Journal* **8** (1965) 250–255.

[14] E.G. Davydov and I.Kh. Sigal, "Application of the penalty function method in integer programming problems", *Engineering Cybernetics* **10** (1972) 21–24.

[15] M. Duran and I.E. Grossmann, "An outer-approximation algorithm for a class of Mixed–Integer Nonlinear Programs", *Mathematical Programming* **36** (1986) 307-339.

[16] M. Duran and I.E. Grossmann, "A mixed–integer programming algorithm for process systems synthesis", *American Institute of Chemical Engineers Journal* **32** (1986) 592–606.

[17] R. Fletcher, *Practical Methods of Optimization, 2^{nd} edition* (John Wiley, Chichester, 1987).

[18] R. Fletcher, "Resolving Degeneracy in Quadratic Programming", University of Dundee, Numerical Analysis Report NA/135 (1991).

[19] R. Fletcher and S. Leyffer, "Solving Mixed Integer Nonlinear Programs by Outer Approximation", University of Dundee, Numerical Analysis Report NA/141 (*to appear in Mathematical Programming*) (1992).

[20] O.E. Flippo A.H.G. Rinnoy Kan and G. van der Hoek, "Duality and decomposition in general mathematical programming", Econometric Institute, Report 8747/B, Erasmus University Rotterdam (1987).

[21] O.E. Flippo and A.H.G. Rinnoy Kan, "A note on Benders Decomposition in mixed–integer quadratic programming", *Operations Research Letters* **9** (1990) 81–83.

[22] O.E. Flippo and A.H.G. Rinnoy Kan, "Decomposition in general mathematical programming", *Mathematical Programming* **60** (1993) 361–382.

[23] O. Forster, *Analysis 2* (Friedrich Vieweg Verlag, Braunschweig, 1984).

[24] R.S. Garfinkel and G.L. Nemhauser, *Integer Programming* (John Wiley, New York, 1972).

[25] A.M. Geoffrion, "Duality in nonlinear programming: A simplified application oriented development", *SIAM Review* **13** (1971), 1–37.

[26] A.M. Geoffrion, "Generalized Benders Decomposition", *Journal of Optimization Theory and Applications* **10** (1972) 237-260.

[27] A.M. Geoffrion, "Lagrangean relaxation for integer programming", *Mathematical Programming Study* **2** (1974) 82–114.

[28] P.E. Gill, W. Murray, M.A. Saunders, J.A. Tomlin and M.H. Wright, "On projected Newton barrier methods for linear programming and an an equivalence to Karmarkar's projective method", *Mathematical Programming* **36** (1986) 183–209.

[29] F. Glover, "Improved linear integer programming formulations of nonlinear integer problems", *Management Science* **22** (1975) 455-460.

[30] F. Glover and D. Sommer, "Pitfalls of rounding in discrete management decision problems", *Decision Science* **22** (1975) 43–50.

[31] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs", *Mathematical Programming* **27** (1983) 1-33.

[32] H. Greenberg, *Integer Programming*, (Academic Press, New York, 1971).

[33] I.E. Grossmann, V.T Voudouris and O. Ghattas, " Mixed–Integer Linear Programming Reformulations for Some Nonlinear Discrete Design Optimization Problems", in *Recent Advances in Global Optimization* (Editors Floudas and Pardalos), (Princeton, 1992).

[34] M. Guignard and S. Kim, "Lagrangean Decomposition: A model yielding stronger Lagrangean bounds", *Mathematical Programming* **39** (1987) 215–228.

[35] O. K. Gupta and A. Ravindran, "Nonlinear Integer Programming and Discrete Optimization", *Transactions of the ASME, Journal of Mechanisms, Transmissions and Automation in Design* **105** (1983) 160–164.

[36] O. K. Gupta and A. Ravindran, "Branch and bound experiments in convex nonlinear integer programming", *Management Science*, **31** (1985) 1533-1546.

[37] P. Hajela and C.-J. Shih, "Optimal design of laminated composites using a modified mixed integer and discrete programming algorithm" *Computers and Structures* **32** (1989) 213-221.

[38] W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, (Springer–Verlag, Berlin, 1981).

[39] R.G. Jeroslow, "There cannot be any algorithm for integer programming with quadratic constraints", *Operations Research* **21** (1973) 221-224.

[40] R.G. Jerroslow, "Trivial integer programs unsolvable by branch–and–bound", *Mathematical Programming* **6** (1974) 105–109.

[41] N. Karmarkar, "A new polynomial–time algorithm for linear programming", *Combinatorics* **4** (1984) 373–395.

[42] G.R. Kocis and I.E. Grossmann, "Global Optimization of Nonconvex Mixed–Integer Nonlinear Programming (MINLP) problems in Process Synthesis", *Industrial Engineering Chemistry Research* **27** (1988) 1407-1421.

[43] G.R. Kocis and I.E. Grossmann, "Computational experience with DICOPT solving MINLP problems in process systems engineering", *Computers and Chemical Engineering* **13(3)** (1989) 307–315.

[44] F. Körner, "Integer quadratic optimization", *European Journal of Operational Research* **19** (1985) 268–273.

[45] F. Körner, "A new branching rule for the branch and bound algorithm for solving nonlinear integer programming problems", *BIT* **28** (1988) 701-708.

[46] J.W. Kwiatowski, "Algorithms for Index Tracking", University of Edinburgh, Centre for Financial Markets Research, Working Paper 90.1 (1990).

[47] A.H. Land and A.G Doig, "An automatic method for solving discrete programming problems", *Econometrica* **28** (1960) 497–520.

[48] R. Lazimy, "Mixed–integer quadratic programming", *Mathematical Programming* **22** (1982) 332–349.

[49] R. Lazimy, "Improved Algorithm for Mixed-Integer Quadratic Programs and a Computational Study", *Mathematical Programming* **32** (1985) 100–113.

[50] Han-Lin Li, "An approximate method for local optima for nonlinear mixed integer programming problems", *Computers and Operations Research* **19** (1992) 435–444.

[51] J.D.C. Little, K.C. Murty, D.M. Sweeney and C. Karel, "An algorithm for the travelling salesman problem", *Operations Research* **11** (1963) 972–989.

[52] Han Tong Loh and P.Y. Papalambros, "A sequential linearization approach for solving mixed-discrete nonlinear design optimization problems", *Transactions of the ASME, Journal of Mechanical Design* **113** (1991) 325–334.

[53] Han Tong Loh and P.Y. Papalambros, "Computational Implementation and tests of a sequential linearization algorithm for mixed–discrete nonlinear design problems", *Transactions of the ASME, Journal of Mechanical Design* **113** (1991) 335–345.

[54] P. Michelon and N. Maculan, "Lagrangean decomposition for integer nonlinear programming with linear constraints", *Mathematical Programming* **52** (1991) 303–313.

[55] *The NAG Fortran Library Manual, Mark 15*, (Oxford, 1991).

[56] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization*, (John Wiley, New York, 1988).

[57] G.R. Olsen and G.N. Vanderplaats, "Method for nonlinear optimization with discrete design variables", *AIAA Journal* **27** (1989) 1584–1589.

[58] M. Padberg and G. Rinaldi, "A branch–and–cut algorithm for the resolution of large-scale symmetric travelling salesman problems", *SIAM Review* **33** (1991) 40-100.

[59] P.M. Pardalos and J.B. Rosen, *Constrained Global Optimization: Algorithms and Applications*, (Springer Verlag, Berlin, 1987).

[60] I. Quesada and I.G.Grossmann, "An LP/NLP based branch–and–bound algorithm for convex MINLP optimization problems", *Dept. of Chemical Engineering, Carnegie Mellon University* (1992).

[61] J.B. Rosen and S. Suzuki, "Construction of nonlinear programming test problems", *Communications of the ACM* **8** (1965) 113.

[62] N.V. Sahinidis and I.E. Grossmann, "Convergence properties of generalized Benders Decomposition", *Computers and chemical Engineering* **15** (1991) 481-491.

[63] R.L. Salcedo, "Solving nonconvex nonlinear programming and mixed–integer nonlinear programming problems with adaptive random search", *Industrial Engineering Chemistry Research* **31** (1992) 262–273.

[64] E. Sandgren, "Nonlinear Integer and Discrete Programming for Topological Decision Making in Engineering Design", *Transactions of the ASME, Journal of Mechanical Design* **112** (1990) 118–122.

[65] E. Sandgren, "Nonlinear Integer and Discrete Programming in Mechanical Design Optimization", *Transactions of the ASME, Journal of Mechanical Design* **112** (1990) 223–229.

[66] H. Schramm and J. Zowe, "A Version of the Bundle Idea for Minimizing a Nonsmooth Function: Conceptual Idea, Convergence Analysis, Numerical Results", *SIAM Journal on Optimization* **2** (1992) 121–152.

[67] A. Schrijver, *Theory of Linear and Integer Programming*, (John Wiley, Chichester, 1986).

[68] U. Stammbach, *Lineare Algebra* (B.G. Teubner, Stuttgart, 1983).

[69] H.A. Taha, *Integer Programming – Theory, Applications and Computations*, (Academic Press, London, 1975).

[70] F.E. Torres, "Linearization of mixed–integer products", *Mathematical Programming* **49** (1991) 427–428.

[71] H. Tuy, "Global minimization of a difference of two convex functions", *Mathematical Programming Study* **30** (1987) 150–182.

[72] T. Van Roy, "Cross Decomposition for mixed integer programming", *Mathematical Programming*, **25** (1983) 46–63.

[73] J. Viswanathan and I.E. Grossmann, "A combined penalty function and outer–approximation method for MINLP optimization", *Computers and chemical Engineering* **14** (1990) 769-782.

[74] J. Viswanathan, "Integer cuts: The general case", *private communication* (1992).

[75] O.V. Volkovich, V.A. Roshchin and I.V. Sergienko, "Models and methods of solution of quadratic integer programming problems", *Cybernetics* **23** (1987) 289-305.

[76] M. Werman and D. Magagnosc, "The relationship between integer and real solutions of constraint convex programming", *Mathematical Programming* **51** (1991) 133-135.

[77] K.P. Wong, "Decentralized planning by vertical decomposition of an economic system: a non–linear approach", Ph.D. Thesis in National Economics Planning, University of Birmingham.

[78] H.P. Williams, *Model Solving in Mathematical Programming* (John Wiley, Chichester, 1993).

[79] X. Yuan, S. Zhang L. Pibouleau and S. Domenech, "Une méthode d'optimization non linéaire en variables mixtes pour la conception de procédés", *Operations Research* **22** (1988) 331-346.