Solving large MINLPs on computational grids*

Jean-Pierre Goux[†] and Sven Leyffer[‡]

24 May 2001

Abstract

We consider the solution of Mixed Integer Nonlinear Programming (MINLP) problems by a parallel implementation of nonlinear branch-and-bound on a computational grid or meta-computer. Computational experience on a set of large MINLPs is reported which indicates that this approach is efficient for the solution of large MINLPs.

Keywords: Mixed Integer Nonlinear Programming, parallel branch-and-bound, computational grid, meta-computing.

AMS 1991 classification: 90C11, 90C30, 49M20.

1 Introduction

We consider the solution of Mixed Integer Nonlinear Programming (MINLP) problems by a parallel implementation of nonlinear branch-and-bound. An important feature of our implementation is the use of a computational grid or meta-computer as the underlying computing platform.

A computational grid is characterized by a collection of loosely-coupled, geographically distributed set of heterogeneous computing resources. For instance, the computing network within a company or a university or several universities and research institutions can be seen as a computational grid. The advantage of using computational grid is that idle workstations can be assembled to solve large computationally challenging problems making it an inexpensive, readily available and powerful parallel machine.

MINLPs are nonlinear optimization problems where some of the variables are required to take integer or discrete values. Problems of this type have many important applications (see Section 1.1) and are conveniently expressed as

$$(P) \begin{cases} \underset{x,y}{\text{minimize}} & f(x,y) \\ \text{subject to} & g(x,y) \leq 0 \\ & x \in X,\ y \in Y \text{ integer.} \end{cases}$$

*Numerical Analysis Report NA/200, Department of Mathematics, University of Dundee.

†`jean-pierre.goux@artelys.com`, Artelys, 215 rue Jean-Jacques Rousseau, 92136 Issy-les-Moulineaux Cedex, France.

‡`sleyffer@maths.dundee.ac.uk`, Department of Mathematics, University of Dundee, DD1 4HN, U.K.

Throughout the paper it is assumed that $X \subset \mathbb{R}^n$ and $Y \subset \mathbb{R}^p$ are compact sets (e.g. polyhedral sets or simple bounds) and that f and g are twice continuously differentiable. These assumptions are quite mild in the sense that most Nonlinear Programming (NLP) solvers require similar assumptions in order to converge to a (local) minimum. In particular, it is not assumed that the continuous relaxation of (P) is a convex programming problem. We will comment on this issue later.

The aim of this paper is to demonstrate that computational grids can assist in the solution of large, computationally challenging MINLP problems. The remainder of the paper is organized as follows: first, some applications of MINLP are reviewed and the computational environment is described, highlighting some of its advantages and difficulties. In Section 2, nonlinear branch-and-bound is discussed together with other solution techniques. Our parallel strategy and implementation details are described in Section 3. Finally, Section 4 presents the numerical experience with the parallel solver on a set of large MINLP applications. This experience indicates the potential of our approach for solving large MINLP applications.

1.1 Applications of MINLP

Mixed Integer Nonlinear Programming problems arise in a wide variety of applications. Chemical engineering applications include process synthesis [38]), batch plant design [31], cyclic scheduling [37] and the design of distillation columns [59]. MINLP problems also arise as nonlinear cutting stock problems in the paper industry [35] and in the optimization of pump configurations [62]. Modeling of a simultaneous model structure and parameter estimation in infrared spectroscopy [55] leads to large convex Mixed Integer Quadratic Programming (MIQP) problems.

Recently, interest in MINLP models and solution techniques has also been motivated by applications from the nuclear industry. The problem here is to maximize the efficiency or performance of a nuclear reactor core after the re-loading operation [52]. A new and interesting area of applications is topology optimization [53] where binary variables model the presence or absence of material in each finite element.

Other MINLP applications arise in the design of water [6] networks, where the direction of flow through a pipe is modeled by a disjunction. MINLPs also arise in financial applications such as strategic planning in telecommunication network design, where integers represent the number of optical fibers to be placed in pipes and nonlinearities arise from the elasticity regarding future pricing strategies [36]. Another important financial application is index tracking for passive portfolio management [39]. MINLPs also arise in the optimization of gas lifted oil well networks [32] and in unit commitment problems in the power generation industry [3].

The survey by Grossmann and Kravanja [30] gives further references. We expect this list to grow significantly in the future due to the availability of modeling languages such as AMPL [22] and GAMS [7] which assist in the formulation of MINLP and enable easy access to solvers. There also exists a growing number of MINLP test problems available on-line, e.g. [44] (where AMPL models of many of the applications referred to here can be found), [24], [21] and the newly developed MINLP-world at [23].

1.2 Computational grids

A computational grid or meta-computer is characterized by a collection of loosely-coupled, geographically distributed and distributively owned set of heterogeneous computing resources. A computational grid enables idle workstations to be assembled to solve large computationally challenging problems such as MINLPs. This makes computational grids an inexpensive, readily available and powerful alternative to traditional supercomputers.

Besides these advantages, the use of computational grids also provides some difficult challenges for the algorithm developer.

1. The machines are *dynamically available* and the pool of available machines may grow and shrink during computation.

2. Individual machines can *become unavailable* at any time while a job is running, for instance when the owner returns to use it.

3. The *communication bandwidth* is potentially very low and the latency very high and may vary over time.

4. The grid is composed of *heterogeneous resources* with different architectures, operating systems, memory sizes, processor speeds etc.

These points have implications for the design and implementation of parallel algorithms on computational grids. The dynamic availability means that the parallel implementation cannot rely on having a fixed amount of resources available. The fact that machines can become unavailable implies that the algorithm has to be fault-tolerant. The low latency has implications for the grain-size and the amount of communication between resources. These issues are addressed in detail in Section 3.

In order to harness the potential of computational grids, two main tools are used in this work. These are Condor [47], a resource manager and MW [28] a framework for building parallel master-worker application, see Section 3.3. Condor manages a pool of distributively owned workstation. It allows the owner of each machine to retain control over the access rights to his machine, e.g. by specifying access hours or conditions under which Condor must terminate a job. When a job is submitted to Condor, it discovers an idle machine and assigns it to the job. Condor also allows memory checkpointing to be done if machines become unavailable, enabling it to transfer a job to another available machine in the pool.

There have been several implementations of parallel MILP branch-and-bound algorithms. Most notably, Eckstein [14] implements a parallel branch-and-bound solver on a CM-5. However, the CM-5 is a dedicated parallel machine and very different from a computational grid. Chen and Ferris [8], see also [9], implement a parallel MILP solver on a computational grid, but their parallel strategy differs slightly from ours. Anstreicher et al. [2] solve some large Quadratic Assignment Problems on a computational grid. Finally, Laursen [41] shows that parallel branch-and-bound without communication between workers can be efficient, provided a good initial solution is known. We believe that there are issues unique to MINLP which ensure that the present approach is of interest.

2 Algorithms for MINLP

In this section we describe branch-and-bound and briefly review other techniques for solving MINLP problems. The issues relating to the parallel strategy are presented in Section 3.

2.1 Branch-and-bound

Branch-and-bound dates back to Land and Doig [40]. The first reference to nonlinear branch-and-bound can be found in Dakin [12] although practical implementations have only recently become available with the advance of nonlinear programming techniques. Branch-and-bound is most conveniently explained in terms of a tree-search.

Initially, all integer restrictions are relaxed and the resulting NLP relaxation is solved. If all integer variables take an integer value at the solution then this solution also solves the MINLP. Usually, some integer variables take a non-integer value. The algorithm then selects one of those integer variables, say y_i with value $\hat{y}_i$, and branches on it. Branching generates two new NLP problems by adding simple bounds $y_i \leq [\hat{y}_i]$ and $y_i \geq [\hat{y}_i] + 1$ respectively to the NLP relaxation (where $[a]$ is the largest integer not greater than a).

One of the two new NLP problems is selected and solved next. If the integer variables take non-integer values then branching is repeated, thus generating a branch-and-bound tree whose nodes correspond to NLP problems and where an edge indicates the addition of a branching bound. If one of the following fathoming rules is satisfied, then no branching is required, the corresponding node has been fully explored (fathomed).

1. An infeasible node is detected. In this case the whole subtree starting at this node is infeasible and the node has been fathomed.

2. An integer feasible node is detected. This provides an upper bound on the optimum of the MINLP; branching is not needed and the node has been fathomed.

3. A lower bound on the NLP solution of a node is greater or equal than the current upper bound. In this case the node is fathomed, since this NLP solution provides a lower bound for all problems in the corresponding sub-tree.

Once a node has been fathomed the algorithm backtracks to another node which has not been fathomed until all nodes are fathomed. These rules only guarantee that the global solution to (P) is found, if each NLP can be solved to global optimality. This is the case, if f and g are convex or have some other structure which ensures that each NLP has a unique minimum value. In the case of (P) being nonconvex, no guarantee can be given that the solution is a global minimum. Nevertheless, we have found branch-and-bound to work well in practice even on nonconvex examples.

Many heuristics exist for selecting a branching variable and for choosing the next problem to be solved after a node has been fathomed (see surveys by Gupta and Ravindran [33] and Volkovich et al. [60]). Linderoth and Savelsbergh [45] study the computational effect of MILP heuristics and some of their results (most notably on pseudo-costs) are relevant to MINLP. Some of these ideas are relevant to the parallel MINLP solver and are described in Section 3.

2.2 Choice of NLP solver for branch-and-bound

In order to design a successful nonlinear branch-and-bound solver, there are certain requirements which the underlying NLP solver must satisfy. It has to be robust even if the problems are degenerate, as it is difficult to recover from numerical failures deep in the branch-and-bound tree. It has to declare an NLP infeasible quickly and reliably. In particular, it should *not* declare feasible nodes infeasible as this might cut off global solutions. Finally, it should have good warm-starting properties that exploit the solution of the parent node.

In our implementation, each node in the tree-search is solved with a Sequential Quadratic Programming solver. These methods date back to [64], [34] and [48], see also [10] for a selection of other references. SQP methods possess very fast local convergence properties. We believe that SQP methods are very well suited for use in nonlinear branch-and-bound. In particular, the solver used in the experiments possesses the following properties which we regard as important for the solution of large MINLP problems.

1. The solver uses a robust QP solver which resolves degeneracy at the QP level. During branching, it is often observed that the linearizations at higher levels in the tree are more degenerate and it is important that the solver does not fail under these conditions, see [15].

2. It detects infeasibility quickly by solving a feasibility problem. Solving a feasibility problem explicitly avoids the need to increase the penalty parameter to infinity before declaring a problem locally infeasible. This makes this approach more robust. However, since the feasibility problem is itself usually an NLP, the solver cannot guarantee to find a global solution, unless the problem or the constraints causing the infeasibility have some convexity properties.

3. Finally, SQP methods have better warm-start capabilities than interior point methods. In our implementation, we start each child node from the primal/dual optimal solution of its parent. Numerical experience with a sequential branch-and-bound code indicates that as a consequence, most nodes can be solved in about 3 SQP iterations [43] (see also the results in Section 4).

In our implementation, we use filterSQP [17] to solve each NLP. This method uses a filter to promote convergence from points which are far from a local solution, see [19], [18]. The use of a filter avoids the need to determine a penalty parameter which may be problematic to determine. Finally, the interface to AMPL [26] allows the use of automatic differentiation to evaluate the first and second derivatives required by the solver.

2.3 Review of solution techniques for MINLP

Various solution techniques for solving MINLP have been proposed in the past. In this paper, we concentrate on deterministic methods rather than heuristic solution techniques, since we believe that NLP based solvers are better suited to handle nonlinear equations and inequalities than heuristic techniques, which have mostly been applied to well structured integer linear problems. See [11] for a survey of heuristic techniques. In this section we briefly discuss the various deterministic solution techniques and motivate the choice of branch-and-bound for parallelization on computational grids.

Alternatives to nonlinear branch-and-bound usually decompose (P) into an alternating sequence of MILP master problems and NLP subproblems. The solution of each NLP provides a number of cuts which are added to the master problem. Outer Approximation, e.g. [13], [65] and [16], Benders Decomposition, e.g. [4], [27] and [20] and the Extended Cutting plane method, e.g. [61] fall into this category. In order to handle nonconvex MINLPs, these methods need to modify the cuts by lowering (and thereby weakening) them, see e.g. [38] for Outer Approximation and [56] for the Extended Cutting plane method. Often, these techniques are only available for special classes of MINLP and do not generalize to more general problems of type (P).

A method which combines Outer Approximation with branch-and-bound is LP/NLP based branch-and-bound, [50] and [42]. Like in Outer Approximation a MILP master program is constructed which is solved using branch-and-bound with the additional caveat that the tree-search is interrupted at each integer feasible node, where a corresponding NLP subproblem is solved, new cuts are then added to the MILP master problem and the tree-search continues. Similar heuristics to the ones used in Outer Approximation are necessary to handle nonconvex problems.

In this paper, we decided to parallelize nonlinear branch-and-bound, since the algorithm can naturally be broken down into (almost) independent subproblems which can be solved simultaneously. Branch-and-bound is also readily mapped to a master-worker paradigm which is well suited for computational grids. Moreover, branch-and-bound requires less synchronization than Outer Approximation or Benders Decomposition, which alternate between the solution of an MILP master and NLP subproblems. Finally, another advantage in parallelizing branch-and-bound is that all tasks are of the same nature, while the alternative approaches require two different kinds of task. Clearly, other solution approaches would have been possible. We hope our experience encourages others to do research in this growing area.

3 Parallel branch-and-bound

This section describes the parallelization strategy as well as the underlying MINLP tree-search strategies. In this work, we have chosen a master-worker paradigm to parallelize branch-and-bound. The master manages a pool of unresolved nodes of the tree, sending nodes to workers. Each worker, in turn searches a (distinct) part of the tree using a sequential nonlinear branch-and-bound algorithm. This underlying sequential solver is described first, before introducing the master-worker paradigm. Finally, we comment on certain implementation issues.

3.1 The underlying sequential MINLP solver

The core of the parallel MINLP solver is a sequential nonlinear branch-and-bound solver. This solver accepts MINLPs written in the AMPL format [22], which provides a flexible syntax to quickly specify MINLP models. The solver supports binary and general integer variables as well as special ordered set variables of type 1. Below, we refer to all these different types of variables simply as integer variables. See [63] for a good description of branching on special ordered set variables.

For each of these integer variables, the user can supply branching priorities to guide the solver in its branching decision. Priorities are often readily available and are an effective way of reducing the tree that has to be searched for many applications. During the tree search, whenever a non-integer feasible node is found, the branching variable is chosen from among the integer variables taking non-integral value with the highest priority.

An implementation of a branch-and-bound solver outlined in Section 2.1 would not be able to solve large MINLPs in a reasonable amount of time and storage. To this end, we have added some advanced features borrowed from MILP to the solver. These include advanced rules for choosing a branching variable and rules for selecting the next node to be solved.

The solver allows three different rules for choosing the branching variable from among the variables with highest priority.

- *Maximal fractional branching:* this rule selects the variable with the maximal integer violation for branching.

- *Strong branching:* this rule evaluates both child nodes for all integer variables with highest priority and, based on this, selects the branching variable which degrades the objective value the most. This rule is computationally expensive but helps to take better branching decisions and can be useful at finding good solutions early on.

- *Pseudo-costs branching:* this rule uses pseudo-costs (e.g. [63]) to estimate which branching decisions will degrade the objective value the most. The pseudo-costs are either initialized by the user, or computed using dual information at the root node or computed explicitly using strong branching.

  The pseudo-costs are updated, each time a new branch is explored. This provides a more global view of pseudo-costs as they are averaged over the entire tree-search.

In order to tackle hard problems, a compromise needs to be found between the quality of the branching decisions and its computational effort. To this purpose the user has the possibility to use mixed branching rules depending on the node's depth.

Once a branching variable has been selected, the solver chooses the next node to be explored. Four strategies for selecting the next node to be solved from among the un-solved nodes have been considered.

- *Depth-first:* this strategy selects the deepest node to be solved next. It is easy to show that this strategy minimizes the number of NLP nodes that have to be stored and therefore minimizes the amount of memory required.

- *Best lower bound:* with this strategy, the node with the least lower bound (value of the parent node) is chosen.

- *Best expected bound:* in this strategy, the node with the best expected bound is selected. The expected bound is computed by adding pseudo-cost degradation estimates of the branching variable to the lower bound.

- *Best estimate (using pseudo-costs):* this strategy chooses the node leading to the best expected integer solution. The value of the best expected integer solution in a subtree can be estimated by adding the pseudo-cost estimates for *all* non-integral integer variables to the lower bound.

The code has been written in C++ to allow easy plug-in of other NLP solvers and addition of features such as cutting planes. The code can also solve MINLPs in sequential mode very effectively and some of the features referred to above also appear in a recent commercial branch-and-bound solver `SBB` [25]. It has been our aim to ensure that the sequential version of our parallel solvers works reasonably efficiently. This ensures that the numerical results of Section 4 give an indication as to the amount of improvement that a commercial MINLP solver could expect from using parallel techniques like the ones described here.

3.2 Parallel strategy

The choice of parallel strategy has been directly influenced by the principal characteristics of computational grids, see Section 1.2. The dynamic availability of resources and the low communication bandwidth made us choose a master-worker paradigm. See [29] for a thorough justification of the suitability of the master-worker paradigm to computational grids. In this paradigm, a master machine manages the task pool. As machines join the computational pool, tasks are sent to them and task results are sent back to the master. When a machine disappears its task is reassigned to the next available machine.

In order to obtain high parallel efficiency, the type of task has to be chosen so as to avoid redundant work, minimize load at the master level and maximize usage of the worker machines. In a first implementation, each worker was assigned a single NLP node. This turned out to be very inefficient as each NLP was solved too fast for the master to handle the communication, resulting in very poor parallel performance.

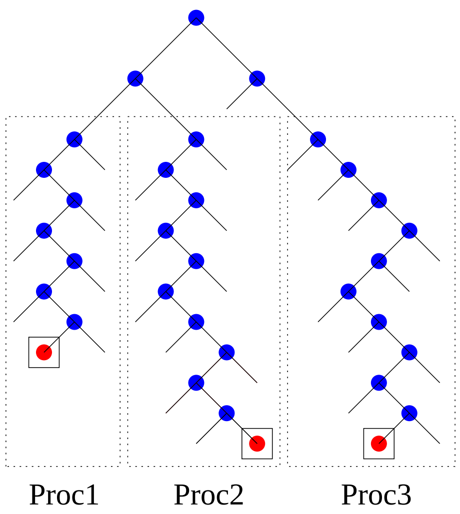

Figure 1: Parallel branch-and-bound strategy

In the current implementation, in contrast, each task is a MINLP subtree rooted at a node sent by the master machine, and this is illustrated in Figure 1. This parallel strategy can be interpreted as a nested tree-search. At the master level, a tree-search is performed by passing subtrees to the workers. Each worker in turn performs a depth-first tree-search on its subtree, thus minimizing the number of tasks returned to the master. All other algorithmic choices (see Section 3.1) are selected by the master. Note that the search strategies at both levels need not be the same.

The work on each subtree must, however, be centrally controlled to a certain degree. Due to the low communication bandwidth of computational grids, there is no further communication between master and worker or between workers until each worker returns the (partial) result of its tree search. This can have undesired consequences, if workers were allowed to search their tree until completion.

1. Once all un-resolved nodes at the master have been sent to workers, no new work is available until at least one worker returns a partially explored tree. This can result in idle times for unemployed workers.

2. A worker may spend a long time exploring a part of the tree that could have been pruned with a new upper bound, thus performing unnecessary work due to a lack of coordination across the grid.

In order to improve load balancing (item 1.) and avoid parallel search anomalies (item 2.), each worker only explores its subtree until

- the subtree has been fully fathomed, or
- an integer feasible solution is found, or
- a time limit has been reached.

The last item requires some additional comments. The time limit has to be small enough to avoid idle workers and long tasks to be lost if workers disappear. In our implementation, this time limit, `max_cpu_time` is 10 seconds if the number of tasks at the master is less than 100 and it is 100 seconds if that number is greater than 100, i.e. once a sufficiently large task pool has been established.

Some preliminary runs still showed a poor parallel efficiency due to the large number of short tasks being returned to the master. To avoid this, the worker's time limit has been relaxed by giving it a recourse period of `max_cpu_time`. In that additional period, the worker tries to eliminate all nodes with a relative gap g_{rel} smaller than a given constant before returning to the master. Here g_{rel} is defined as

$$g_{rel} = \frac{U - f}{U - \hat{f}},$$

where U is the current upper bound, f is the lower bound on the node and $\hat{f}$ is the lower bound on the root node of the subtree. If g_{rel} is close to 1, then the corresponding subtree is deemed to be sufficiently large. During the recourse period, the solver attempts to fathom all nodes with a relative gap, $g_{rel} \leq 0.5$, i.e. nodes with a small expected subtree are fathomed. This results in an increase of the average task size returned to the master and improves the performance significantly.

The success of branch-and-bound depends crucially on the availability of good integer feasible solutions which provide upper bounds and increase the amount of fathoming. In order to find good solutions early on, a *lazy-best-first* strategy has been implemented which corresponds to best-expected-bound node selection at the master level. In this strategy, workers are given the node with the best expected bound in the master task pool. The advantage of this strategy is that early on, nodes are chosen from near the root of the tree, resulting in a more global search of the tree. The drawback of this approach is that it leads to more difficult tasks, which can normally not be completed in `max_cpu_time` seconds. In order to control the size of the master task pool, the lazy-best-first strategy is only pursued until the task pool reaches `max_lazy_best_first`. The master then switches back to depth-first-search until the number of problems in the task pool drops below `min_lazy_best_first`. Both constants depend on the size of problem and the amount of storage available at the master. The effect of the lazy-best-first strategy can be observed in the numerical results of trimlon7, see Figure 3 and the paragraph following that figure.

These issues related to the parallel implementation of a branch-and-bound solver on a computational grid platform have also been used effectively in [2].

3.3 Implementation details

In order to use computational grid resources, resource management software is needed to identify machines joining or leaving the computational pool, assigning jobs to machines and running these jobs remotely. The Condor system [47] developed at the University of Wisconsin Madison provides such features. Condor is a scavenger system that detects idle machines in a network of participating machines, potentially in different geographical locations. The owner of each machine defines the usage/sharing policy and a Condor manager machine keeps track of the available resources based on the resource's status and policies at all time. The Condor manager also takes care of the matchmaking between batch jobs submitted by Condor users and available machines.

The batch submission mode of operation is not effective for implementing parallel algorithms with small granularity like branch-and-bound. Fortunately Condor provides full support of the PVM message passing library [49] and remote I/O (input/output) that can be used to maintain a state in participating resources and make them communicate with each other.

Even with these features there are difficult computer science issues in implementing an algorithm using the master-worker paradigm in a robust and efficient way. The MW software framework ([29] and [28]) is an abstract software framework designed to easily build master-worker applications running on top of Condor. By handling automatically all the hard issues of resource discovery and management, fault-tolerance, task scheduling and interprocess communication, users can focus on the algorithmic features of their application without worrying about the computational details.

Three abstract classes must be implemented for each specific applications. The `MWDriver` class corresponds to the master process and contains the control center for distributing tasks to workers. The `MWTask` class describes the inputs and outputs (problem data and results) that are associated with a single unit of work. The `MWWorker` class contains code to initialize a worker process and to execute any tasks that are sent to it by the master. For a complete description of the functionalities of MW, see [29] and [28].

In our parallel MINLP application, the MINLP master is in charge of reading the MINLP problem in AMPL format, reading the algorithmic parameters supplied by the user, placing the MINLP root node in the task pool and shipping initial information to MINLP workers when they join the computation. During the course of the computation the MINLP master has to

- manage the task pool and the upper level branch-and-bound tree-search,

- merge and store pseudo-costs information and run time statistics from the workers,
- choose the next node to be sent to a worker by alternating node selection strategies to balance load and avoid contention,
- update and store the current best upper and lower bounds, and
- check the stopping criterion.

Each task, or unit of work corresponds to a MINLP subtree. The worker is essentially the MINLP sequential code described in Section 3.1 plugged into the MW framework. The description of the MINLP problem (P) (i.e. information about bounds, function and derivative evaluation) is only passed once to each worker, namely when it joins the task pool. After this initial setup, each worker only received incremental information to describe the subtree that it is being sent. This description includes

- the lower and upper bounds of the root node of the subtree,
- information for warm-starting the search. This includes initial guesses of the primal and dual variables and the trust region radius,
- the most recent pseudo-costs table, and
- the best upper bound found so far by all workers which is used as the cut-off value.

After completing the task (either because the time limit has been reached, or an integer feasible solution has been found or because the subtree has been fathomed), the worker returns the following information to the master

- an integer feasible solution (if any has been found),
- the remaining unexplored nodes which form the remaining task pool,
- pseudo-costs updates and cumulated run-time statistics.

In order to evaluate the performance of an application in a heterogeneous environment, MW provides a feature to register a *benchmark* task that is run by all new MINLP workers as they join the computational pool. In our application this is a small MINLP tree search. The CPU time required to process this task is used to normalize the statistics returned by each individual workers and provide a performance evaluation that is independent of the heterogeneous nature of resources, see [28].

In order to ensure robustness of the parallel MINLP solver, the master machine (which can become unavailable like any other resource) is checkpointed occasionally. This means that the list of tasks and all global statistics and pseudo-costs are stored on disk. This enables the solver to continue from the last checkpoint if the master machine becomes unavailable. This problem specific method of checkpointing is far more efficient than the standard Condor checkpointing which consists of writing a complete copy of the current memory status of a machine to disk.

There is no problem specific checkpointing at the worker level. Thus we do not perform checkpointing at the worker level, as the amount of data that would have to be saved is huge. Checkpointing of workers is only useful in applications with much larger granularity (e.g. 1 hour per task). Instead, if a worker becomes unavailable, its task is simply re-scheduled on another worker by the master.

MW has been used to implement other efficient grid-enabled parallel numerical optimization solvers, such as solver for stochastic optimization [46], the quadratic assignment problem [2] and mixed-integer linear programming [9].

4 Numerical Experience

This section presents some preliminary numerical results with a parallel implementation of branch-and-bound on a computational grid. The experiments were conducted on a grid of 146 Intel/Solaris machines at the University of Wisconsin's Condor pool. All problems were solved to an absolute tolerance of 10^{-6}.

header	description
name	Name of problem
n	Number of variables (binary, integer & continuous)
n_b	Number of binary variables
n_i	Number of general integer variables
m	Number of constraints
aver. QPs	Average number of QPs per node (excluding root node)
tree-size	Base-10-logarithm of size of the complete tree
B&B-size	Base-10-logarithm of the number of nodes solved
B&B-eff	= B&B-size/tree-size; proportion of tree explored by solver
MINLP	Number of MINLP-subproblem tasks solved
NLP	Number of NLPs solved
inf NLP	Number of infeasible NLPs
job-time	Overall (wall) time for job [in hours]
work-time	Overall (CPU) time for workers [in hours]
work	Average number of workers
effic	Parallel efficiency

Table 1: Description of headers for Tables 2 and 3

Table 1 describes the headers of the two subsequent tables. Table 2 shows the problem characteristics and Table 3 shows the results of the runs. The columns headed "tree-size" and "B&B-size" indicate that the MINLPs attempted here are non-trivial and provide a measure for the efficiency of the branch-and-bound solver. The parallel efficiency is defined as

$$\text{effic} := \frac{\text{work-time}}{\text{work} * \text{job-time}}$$

or the total CPU time of the workers as a proportion of the computational resources used in the solve. This number shows how efficiently the computational grid is exploited by the algorithm. Note that "speed-up" would not be a useful measure in the context of computational grids. In the present context, workstations can disappear at any time and it is impossible to create conditions which would measure speed-up. More importantly though the main aim of computational grids is to facilitate high-throughput computing, exploiting unused cycles. Therefore, parallel efficiency is a better guide to the efficiency of an algorithm.

No.	name	n	m	n_b	n_i	aver. QPs	tree-size	B&B-size	B&B-eff
1.	c-reld-14a	342	308	168	0	1.15	50.6	2.7	10^{-48}
2.	c-reld-q-25	1033	658	625	0	18.58	188.1	2.6	10^{-186}
3.	c-sched2	400	137	308	0	4.91	92.7	4.4	10^{-88}
4.	space-25	893	235	750	0	1.15	225.8	3.8	10^{-222}
5.	stockcycle	480	97	432	48	2.05	235.0	5.6	10^{-229}
6.	trimlon4	24	24	4	20	1.24	18.1	4.6	10^{-14}
7.	trimlon5	35	35	5	30	1.20	27.0	5.8	10^{-21}
8.	trimlon6	48	36	6	42	1.89	37.2	6.1	10^{-31}
9.	trimlon7	63	42	7	56	1.14	51.9	8.9	10^{-43}
10.	trimloss4	105	64	85	0	2.10	25.6	8.0	10^{-18}

Table 2: Problem characteristics

The problems arise in a variety of applications. Problems 1 and 2 are core reload pattern optimization problems (see Quist et al. [52] and [51]). Problem 3 is a cyclic scheduling due to Jain and Grossmann [37]. Problem 4 is a model of a 25 bar space truss design in which the weight of the structure is minimized subject to choosing each bar from a discrete set. This model is due to Tin-Loi [58]. Problem 4 is a pure integer NLP which minimizes the total average stock cycle [54]. Problems 6 to 9 are nonconvex MINLP arising from trim loss minimization in the paper industry. The problems differ in the number of product rolls and this formulation is due to Harjunkoski et al. [35]. Finally, problem 10 is a convexified version of Problem 6. AMPL models describing all problems can be found on-line at [44].

name	MINLP	NLP	inf NLP	job-time	work-time	work	effic
c-reld-14a	17	507	16	0.18	0.11	1.3	45.8
c-reld-q-25	255	337	19	1.83	19.77	17.1	68.3
c-sched2	1094	25112	488	0.46	2.48	25.1	28.6
space-25	678	4452	1182	0.39	0.46	29.7	6.4
stockcycle	2848	384358	2319	1.20	20.67	33.3	57.2
trimlon4	108	28888	8473	0.11	0.06	3.0	49.3
trimlon5	283	556457	145764	0.14	0.42	13.3	22.6
trimlon6	725	1001705	200608	0.20	1.90	22.8	43.2
trimlon7	96408	600518018	179504387	15.50	752.65	62.7	80.5
trimloss4	42205	61291678	28054745	4.88	321.52	93.0	74.7

Table 3: Results of MINLP test

The final two columns of Table 2 show that the branch-and-bound techniques used in the present implementation are efficient for solving MINLP problems. Note that only a very small proportion of the complete tree is explored by the solver. Consider trimlon7, for example: even though it requires the solution of almost 10^9 NLPs, that number represents only a 10^{-43} part of the total tree.

Table 2 shows the success of the warm start techniques in our NLP solver. For most problems, 1-2 QPs are required on average to solve a node in the tree. Only problems "c-sched2" and "c-reld-q-25" have a larger number of QPs per node. The much larger average number of QPs for "c-reld-q-25" explains the high efficiency (Table 3) for this problem, despite its small MINLP tree. These excellent warm start properties of the NLP solver explain the poor parallel efficiency of an earlier implementation in which each worker solved an NLP. Clearly, any such approach would create a bottleneck at the master level, as the NLPs are solved too quickly.

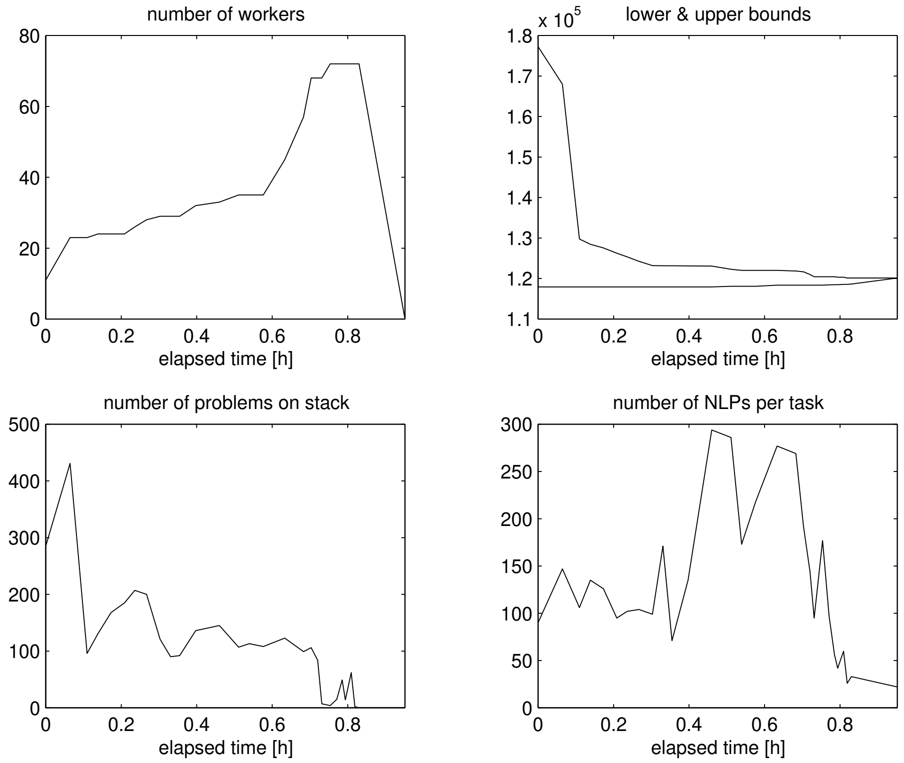

Figure 2: Performance of parallel solver on stockcycle

Figures 2, 3 and 4 show the runs of stockcycle, trimlon7 and trimloss4 respectively in more detail. The figures show the number of workers, the lower and upper bound, the number of problems on the stack of the master and the number of NLPs solved per MINLP task as a function of computing time on the master.

Table 3 shows that the approach presented here is efficient in solving some large MINLP. The efficiency reaches 80% and is well over 40 % for most problems. The efficiency is proportional to the number of MINLP tasks that are solved, except for c-reld-*, where the high efficiency is due to the fact that each NLP is much harder to solve (as can be seen by comparing the number of QPs per node in Table 2).

The running time for each MINLP is reduced by up to two orders of magnitude, making these MINLPs solvable in a reasonable amount of time. Another encouraging result is the fact that the MINLP master is capable of handling huge numbers of MINLP and NLP subproblems without a degradation in efficiency. That is particularly apparent

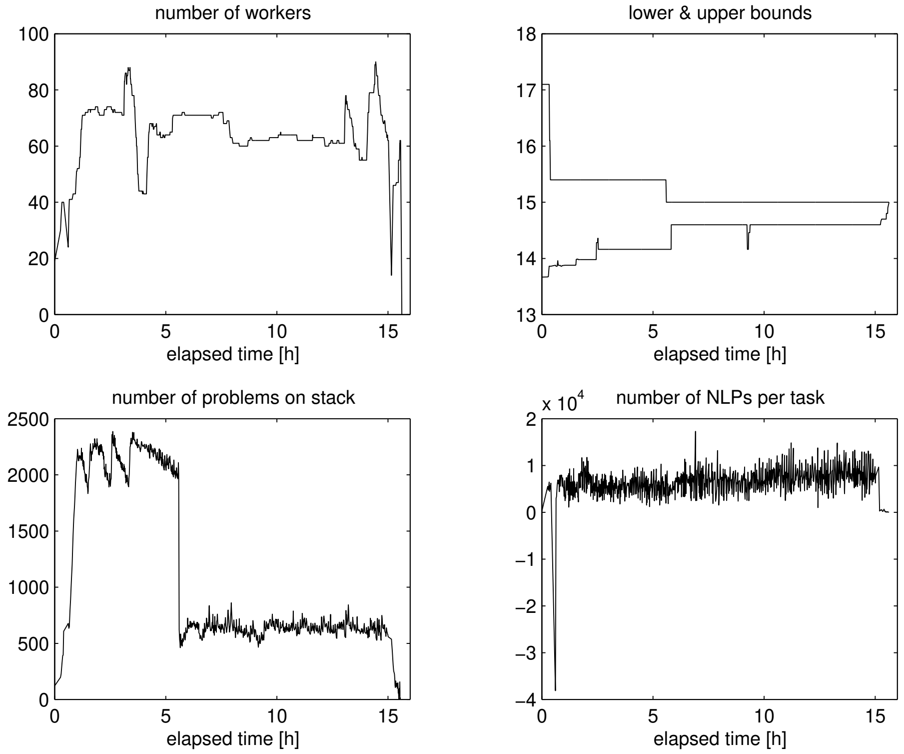

Figure 3: Performance of parallel solver on trimlon7

for trimlon7. This proves in our view that the concept of a master-worker paradigm with each worker solving MINLP problems is useful for tackling challenging MINLP problems.

For the smaller problems (1-8), only a moderate *average* number of workers have been used. This is explained by the relative small number of MINLP tasks which need to be carried out. These tasks only become available dynamically as new nodes are returned to the task pool from the workers. Thus, the solver builds up a pool of workers dynamically, as new tasks are required. This can be seen well in Figure 2, where the number of workers increases. Once, the solver reaches 70 workers, the problem is already solved which explains the moderate average number of workers.

The two largest problems (9 and 10) each require a considerable amount of computational resources as can be seen from Table 3 and Figures 3 and 4. In both cases, the solver rapidly requests idle machines as the task pool increases in size. As machines become unavailable at times, the number of workers falls. Note that the number of workers varies considerably during each of the runs. This is handled automatically by the MW framework.

It can be observed in Figure 3 that the behaviour for the first 5 hours differs from the remainder. This illustrates the effect of the lazy-best-first strategy. Initially, lazy-best-first is performed until the task pool at the master exceeds `max_lazy_best_first`, which is 2100 in this run. At that point, the master resorts to depth-first-search, resulting in a reduction in the number of nodes at the master. Once this number drops below `min_lazy_best_first` (1900 here), lazy-best-first resumes. This results in the number of problems oscillating around 2000 for this run until the first integer solution is found and a large proportion of problems are removed from the tree. The remainder of the time is

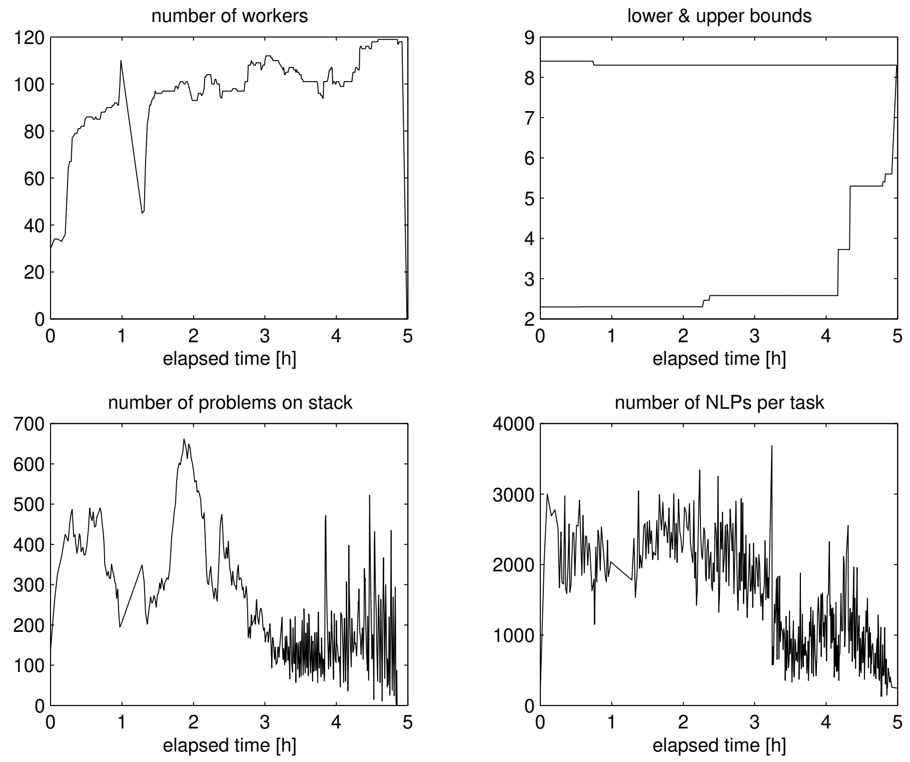

Figure 4: Performance of parallel solver on trimloss4

spend verifying this solution by pruning the remainder of the tree.

An interesting point about Figure 3 is that the lower bound is *not* monotonic. The reasons for this is that problem 9 is *nonconvex*. Thus the bounds cannot be expected to be monotone (in fact the lower bound is not strictly speaking a bound at all).

It can be seen from both Figure 3 and 4 that the size of each task remains of roughly equal size. This is important, as it ensures that the grain size of the tasks is large enough so as not to be dominated by communication costs and over-heads. This is also reflected in the efficiency of the solves.

5 Conclusions

We have presented a parallel implementation of nonlinear branch-and-bound on a computational grid. The parallelization strategy employs a master-worker paradigm in which the master manages a pool of MINLP tasks corresponding to subtrees of the branch-and-bound tree. Each worker solves a MINLP task by nonlinear branch-and-bound.

Preliminary numerical experience demonstrates the efficiency of this approach. The master is able to manage large numbers of MINLP tasks without any drop in parallel efficiency. Many hard MINLP problems are solved up to 2 orders of magnitude faster than a serial implementation could achieve. The approach is efficient both at finding optimal solutions and at verifying their optimality. This makes this approach suitable for solving computationally challenging MINLP problems.

There are various ways in which the current approach can be improved. For instance, it is possible to improve the branch-and-bound solver at the worker level by using early branching [5]. This has been shown to improve on branch-and-bound by a factor of 2-3 [43] albeit at the cost of being less reliable at finding global solutions for nonconvex MINLPs.

Another way to improve the worker's branch-and-bound solver would be to derive cutting planes, e.g. [57] and [1] and to use these in a branch-and-cut framework. Note that, the validity of these cuts depends crucially on the convexity of f and g and it is not clear that nonconvex MINLPs can be solved satisfactorily with this approach. However, it would be interesting to explore the generation of cuts in parallel, possibly by having a small number of workers dedicated to the generation of strong cuts.

Acknowledgements

We are grateful to Jeff Linderoth and Jorge Nocedal for many fruitful discussions and insightful comments on integer programming. Part of this research was carried out while the second author was visiting the stimulating research environment of Argonne National Laboratory and Northwestern University. The support of both institutions is gratefully acknowledged. This work has been supported by National Science Foundation grant CDA-972638 and Engineering and Physical Sciences Research Council grant GR/M59549.

References

[1] Akrotirianakis, I., Maros, I. and Rustem, B. An outer approximation based branch-and-cut algorithm for convex 0-1 MINLP problems. Technical report 2000-06, Department of Computing, Imperial College, London, June 2000.

[2] Anstreicher, K., Brixius, N., Goux, J.P. and Linderoth, J. Solving large quadratic assignment problems on computational grids. Technical report, Argonne National Laboratory, 2000. Available at `www-unix.mcs.anl.gov/metaneos/papers/qap.ps`, to appear in Mathematical Programming.

[3] Batut, J. and Renaud, A. Daily generation scheduling optimization with transmission constraints: A new class of algorithms. *IEEE Transactions on Power Systems*, 7(3):982–989, August 1992.

[4] Benders, J.F. Partitioning procedures for solving mixed-variable programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[5] Borchers, B. and Mitchell, J.E. An improved branch and bound algorithm for Mixed Integer Nonlinear Programming. *Computers and Operations Research*, 21(4):359–367, 1994.

[6] Brooke, A., Drud, A. and Meeraus, A. Modeling systems and nonlinear programming in a research environment. In Ragavan, R. and Rohde, S. M., editors, *Computers in engineering 1985*. 1985. GAMS model at `www.gams.com/modlib/libhtml/waterx.htm`.

[7] Brooke, A., Kendrick, D., Meeraus, A. and Raman, R. *GAMS A user's guide*. GAMS Developments Corporation, 1217 Potomac Street, N.W., Washington DC 20007, USA, December 1998.

[8] Chen, Q. and Ferris, M. FATCOP: A fault tolerant condor-PVM mixed integer program solver. Technical report, University of Wisconsin, 1999.

[9] Chen, Q., Ferris, M. and Linderoth, J. FATCOP 2.0: advanced features in an opportunistic mixed integer programming solver. Technical Report Data Mining Institute 99-11, Computer Sciences Department, University of Wisconsin at Madison, USA, 1999. To appear in Annals of Operations Research.

[10] Conn, A.R., Gould, N.I.M. and Toint, Ph.L. Methods for nonlinear constraints in optimization calculations. Report RAL–TR–96–042, Rutherford Apleton Laboratory, 1996. (To appear in "The State of the Art in Numerical Analysis", I. Duff and G.A. Watson (eds.)).

[11] Corne, D., Dorigo, M. and Glover, F. *New Ideas in Optimization*. McGraw-Hill, 1999.

[12] Dakin, R.J. A tree search algorithm for mixed integer programming problems. *Computer Journal*, 8:250–255, 1965.

[13] Duran, M. and Grossmann, I.E. An outer-approximation algorithm for a class of mixed–integer nonlinear programs. *Mathematical Programming*, 36:307–339, 1986.

[14] Eckstein, J. Parallel branch–and–bound algorithms for general mixed integer programming on the CM–5. *Siam Journal on Optimization*, 4(4):794–814, 1994.

[15] Fletcher, R. Stable reduced Hessian updates for indefinite quadratic programming. Numerical Analysis Report NA/187, University of Dundee, Department of Mathematics, Scotland, UK, January 1999.

[16] Fletcher, R. and Leyffer, S. Solving mixed integer nonlinear programs by outer approximation. *Mathematical Programming*, 66:327–349, 1994.

[17] Fletcher, R. and Leyffer, S. User manual for filterSQP. Numerical Analysis Report NA/181, Dundee University, April 1998.

[18] Fletcher, R., Gould, N.I.M., Leyffer, S. and Toint, Ph.L. Global convergence of trust-region SQP-filter algorithms for general nonlinear programming. Report 99/03, Department of Mathematics, University of Namur, 61, rue de Bruxelles, B-5000 Belgium, November 1999.

[19] Fletcher, R., Leyffer, S. and Toint, Ph.L. On the global convergence of an SLP-filter algorithm. Numerical Analysis Report NA/183, University of Dundee, UK, August 1998.

[20] Flippo, O.E. and Rinnoy Kan, A.H.G. Decomposition in general mathematical programming. *Mathematical Programming*, 60:361–382, 1993.

[21] Floudas, C.A., Pardalos, P.A., Adjiman, C.S., Esposito, W.R., Gumus, Z.H., Harding, S.T., Klepeis, J.L., Meyer, C.A. and Schweiger, C.A. *Handbook of Test Problems in Local and Global Optimization*. Nonconvex Optimization and its Applications. Kluwer Academic Publishers, Dordrecht, June 1999. Test problems in GAMS available at `titan.princeton.edu/TestProblems/`.

[22] Fourer, R., Gay, D.M. and Kernighan, B.W. *AMPL: A modelling Language for Mathematical Programming.* boyd & fraser publishing company, Massachusetts, 1993.

[23] GAMS. GAMS MINLP world. www-page, `http://www.gamsworld.org/minlp/`, 2000.

[24] GAMS. The GAMS model library index. Technical report, `www.gams.com/modlib/modlib.htm`, 2000.

[25] GAMS. *GAMS/SBB user notes*, March 2001.

[26] Gay, D.M. Automatic differentiation of nonlinear AMPL models. In Griewank, A. and Corliss, G.F., editors, *Automatic Differentiation of Algorithms: Theory, Implementation, and Application*, SIAM Proceedings, pages 61–73, Philadelphia, 1991. SIAM.

[27] Geoffrion, A.M. Generalized Benders decomposition. *Journal of Optimization Theory and Applications*, 10:237–260, 1972.

[28] Goux, J.-P., Kulkarni, S., Linderoth, J. and Yoder, M. An enabling framework for master-worker applications on the computation grid. In *Cluster Computing*, pages 43–50, 2000. Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9).

[29] Goux, J.-P., Linderoth, J. and Yoder, M. Metacomputing and the master-worker paradigm. Preprint MCS/ANL P792-0200, Computer Science Division, Argonne National Laboratory, USA, February 2000.

[30] Grossmann, I.E. and Kravanja, Z. Mixed–integer nonlinear programming: A survey of algorithms and applications. In A.R. Conn L.T. Biegler, T.F. Coleman and F.N. Santosa, editors, *Large–Scale Optimization with Applications, Part II: Optimal Design and Control*, New York, Berlin, 1997. Springer.

[31] Grossmann, I.E. and Sargent, R.W.H. Optimal design of multipurpose batch plants. *Ind. Engng. Chem. Process Des. Dev.*, 18:343–348, 1979.

[32] Grothey, A. and McKinnon, K.I.M. Decomposing the optimization of a gas lifted oil well network. Technical Report MS 00-005, Department of Mathematics and Statistics, University of Edinburgh, Edinburgh, EH9 3JZ, UK, March 2000.

[33] Gupta, O.K. and Ravindran, A. Branch and bound experiments in convex nonlinear integer programming. *Management Science*, 31:1533–1546, 1985.

[34] Han, S.P. A globally convergent method for nonlinear programming. *Journal of Optimization Theory and Applications*, 22(3):297–309, 1977.

[35] Harjunkoski, I., Westerlund, T., Pörn, R. and Skrifvars, H. Different transformations for solving non-convex trim-loss problems by MINLP. *European Journal of Operational Research*, 105:594–603, 1998.

[36] Horner, P. Something to crowe about. *OR/MS today*, pages 38–44, June 2000.

[37] Jain, V. and Grossmann, I.E. Cyclic scheduling of continuous parallel-process units with decaying performance. *AIChE Journal*, 44:1623–1636, 1998.

[38] Kocis, G.R. and Grossmann, I.E. Global optimization of nonconvex mixed–integer nonlinear programming (MINLP) problems in process synthesis. *Industrial Engineering Chemistry Research*, 27:1407–1421, 1988.

[39] Kwiatowski, J.W. Algorithms for index tracking. Working Paper 90.1, University of Edinburgh, Centre for Financial Markets Research, 1990.

[40] Land, A.H. and Doig, A.G. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[41] Laursen, P.S. Can parallel branch and bound without communication be effective? *SIAM J. Optimization*, 4(2):288–296, 1994.

[42] Leyffer, S. Generalized outer approximation. Numerical Analysis Report NA/178, University of Dundee, Dundee, UK, December 1997. (to appear in Encyclopedia of Optimization, editors C.M. Floudas and P.M.Pardalos, Kluwer Academic).

[43] Leyffer, S. Integrating SQP and branch-and-bound for mixed integer nonlinear programming. Numerical Analysis Report NA/182, University of Dundee, UK, August 1998. (to appear in Computational Optimization & Applications).

[44] Leyffer, S. MacMINLP: AMPL collection of mixed integer nonlinear programs. Technical report, University of Dundee, `www.maths.dundee.ac.uk/~sleyffer/MacMINLP/`, July 2000.

[45] Linderoth, J.T. and Savelsbergh, M.W.P. A computational study of search strategies for mixed integer programming. *INFORMS Journal of Computing*, 11:173–187, 1998.

[46] Linderoth, J.T. and Wright, S.J. Decomposition algorithms for stochastic programming on a computational grid. Preprint P875-0401, MCS Division, Argonne National Laboratory, USA, April 2001.

[47] Livny, M., Basney, J., Raman, R. and Tannenbaum, T. Mechanisms for high-throughput computing. In *Speedup 11*, 1997. Available from `http://www.cs.wisc.edu/condor/doc/htc_mech.ps`.

[48] Powell, M.J.D. A fast algorithm for nonlinearly constrained optimization calculations. In G.A. Watson, editor, *Numerical Analysis, 1977*, pages 144–157, Berlin, 1978. Springer–Verlag.

[49] Pruyne, J. and Livny, M. Interfacing condor and PVM to harness the cycles of workstation clusters. *Journal on Future Generations of Computer Systems*, 12:53–65, 1996.

[50] Quesada, I. and Grossmann, I.E. An LP/NLP based branch–and–bound algorithm for convex MINLP optimization problems. *Computers and Chemical Engineering*, 16:937–947, 1992.

[51] Quist, A.J. *Application of Mathematical Optimization Techniques to Nuclear Reactor Reload Pattern Design.* PhD thesis, Technische Universiteit Delft, Thomas Stieltjes Institute for Mathematics, The Netherlands, 2000.

[52] Quist, A.J., van Geemert, R., Hoogenboom, J.E., Illés, T., de Klerk, E., Roos, C. and Terlaky, T. Optimization of a nuclear reactor core reload pattern using nonlinear optimization and search heuristics. Draft paper, Delft University of Technology, Faculty of Applied Mathematics, Department of Operation Research, Mekelweg 4, 2628 CD Delft, The Netherlands, September 1997.

[53] Sigmund, O. A 99 line topology optimization code written in matlab. *Structural Optimization*, 2000. (to appear).

[54] Silver, E.A. and Moon, I. A fast heuristic for minimizing total average cycle stock subject to practical constraints. *Journal Operations Research Society*, 50:789–796, August 1999.

[55] Skrifvars, H., Leyffer, S. and Westerlund, T. Comparison of certain MINLP algorithms when applied to a model structure determination and parameter estimation problem. *Computers & Chemical Engineering*, 22(11), 1998.

[56] Still, C. and Westerlund, T. Convergence properties of the α–ECP method – an algorithm for solving quasi–convex MINLP problems. Manuscript, Process Design Laboratory, Abo Akademi, Abo, Finland, October 1996. (possible to construct a counter–example).

[57] Stubbs, R.A. and Mehrotra, S. A branch–and–cut method for 0–1 mixed convex programming. *Mathematical Programming*, 86:515–532, 1999.

[58] Tin-Loi, F. Modelling bar space truss design as MINLPs. Private communication, University of New South Wales, April 2000.

[59] Viswanathan, J. and Grossmann, I.E. Optimal feed location and number of trays for distillation columns with multiple feeds. *I&EC Research*, 32:2942–2949, 1993.

[60] Volkovich, O.V., Roshchin, V.A. and Sergienko, I.V. Models and methods of solution of quadratic integer programming problems. *Cybernetics*, 23:289–305, 1987.

[61] Westerlund, T. and Pettersson, F. An extended cutting plane method for solving convex MINLP problems. *Computers and Chemical Engineering Supplement*, 19:S131–S136, 1995. ESCAPE–95.

[62] Westerlund, T., Pettersson, F. and Grossmann, I.E. Optimization of pump configurations as MINLP problem. *Computers & Chemical Engineering*, 18(9):845–858, 1994.

[63] Williams, H.P. *Model Solving in Mathematical Programming.* John Wiley & Sons Ltd., Chichester, 1993.

[64] Wilson, R.B. *A simplicial algorithm for concave programming.* PhD thesis, Harvard University Graduate School of Business Administration, 1963.

[65] Yuan, X., Zhang, S., Pibouleau, L. and Domenech, S. Une méthode d'optimisation non linéaire en variables mixtes pour la conception de procédés. *Operations Research*, 22:331–346, 1988.