INFORMS Journal on Computing
Articles in Advance, pp. 1–10
ISSN 1091-9856 | EISSN 1526-5528

DOI 10.1287/ijoc.1080.0293

Solving Hard Mixed-Integer Programming Problems with Xpress-MP: A MIPLIB 2003 Case Study

Richard Laundy, Michael Perregaard
Fair Isaac, Leamington Spa, Warwicks CV32 5YN, United Kingdom
{richardlaundy@fairisaac.com, michaelperregaard@fairisaac.com}

Gabriel Tavares, Horia Tipi, Alkis Vazacopoulos
Fair Isaac, Englewood Cliffs, New Jersey 07632
{gabrieltavares@fairisaac.com, horiatipi@fairisaac.com, alkisvazacopoulos@fairisaac.com}

Despite the fact that no polynomial-time algorithm is known for solving mixed-integer programming (MIP) problems, there has been remarkable success in recent years in solving a wide range of difficult MIPs. In this paper, we take a look at some of the hardest problems in the MIPLIB 2003 test set and show how Xpress-MP can be used to solve some of the problems that were previously thought to be intractable.

Key words: mixed-integer programming (MIP); MIPLIB 2003; software for MIP
History: Accepted by John Chinneck, former Area Editor for Modeling: Methods and Analysis; received February 2007; revised October 2007; accepted May 2008. Published online in *Articles in Advance*.

1. Introduction

Over the past decade there has been a huge increase in the performance of state-of-the-art mixed-integer programming (MIP) codes. This improvement in performance, combined with the increased performance of computers, has dramatically increased the size and complexity of problems that can be solved. As a result, MIP is now commonly used in decision-oriented applications across a wide number of industries.

Xpress-MP is a suite of mathematical modeling and optimization tools used to solve linear, integer, quadratic, nonlinear, and stochastic programming problems (Ashford 2007; Dash Optimization 2005, 2008; Guéret et al. 2002). The Xpress-MP suite is available on most computer platforms and in different capacities for solving problems of various sizes.

Xpress-MP was first released in 1983 as a tool for modeling and solving linear programming (LP) models on PCs, and it was extended in 1986 to solve MIP models by creating a linear programming branch-and-bound code (see Wolsey 1998). Since then, the code has become increasingly more sophisticated with the addition of new search strategies and solution techniques. Some changes fundamental to the improvement in MIP performance are

- robust high-performance implementations of the underlying LP solver, which are up to 100 times faster than implementations of 20 years ago on the same hardware;
- automatically generated cutting planes applied both at the root and in the branch-and-bound tree;
- strong branching; and
- heuristics.

Some of the innovations in MIP solution technology that have been implemented in Xpress have resulted in orders-of-magnitude improvements in solution times, so that problems that were previously considered to be insoluble are now solved routinely within a short period of time. Some of these novel ideas are only applicable to certain problem classes, but their presence in a large toolbox of techniques allows Xpress to solve a wide range of problems derived from a large number of practical problems.

As a result of the vast improvements of the MIP technology within Xpress-MP, we decided that it was time to revisit some of the harder problems in MIPLIB 2003 to see what can be achieved. We show how the toolbox of techniques within Xpress can be harnessed to finally solve some of the problems that have seemed to be intractable in the past.

2. MIPLIB 2003

MIPLIB 2003 is a standard and widely used benchmark to compare the performance of various MIP algorithms.

This library is publicly available at http://miplib.zib.de (Achterberg et al. 2006b) and consists of 60 (minimization) problems, each one having special characteristics, and described in more detail in Achterberg et al. (2006a).

The problems in the MIPLIB test set come from a wide range of applications and range from very

easy problems to problems for which even solving the LP relaxation is a challenging task. According to the MIPLIB website (visited during October 2006), the MIPLIB 2003 instances were classified into three groups of difficulty as follows:

- 28 problems are solved within one hour with a commercial solver;
- 18 problems have their optimal solution known, but they do not satisfy the previous conditions; and
- 14 problems are unsolved.

When we started our investigation, there was still no known solution to one of problems, *stp3d*, and 11 of the 14 unsolved problems were still unsolved. Solutions to three of the problems had just been found by Ferris (2006), who reported the solution of problems *a1c1s1* and *timtab2* using GAMS Grid computing in Condor and showed that the traveling salesman problem *swath* could be solved quickly by applying subtour elimination cuts.

In about a one-year period of time prior to this report, the number of open problems from MIPLIB decreased from 17 to 8 unsolved cases:

December 2005: A solution of *arki001* was proven to be optimal by Balas and Saxena (2005) using the split closure of the problem.

June 2006: According to the MIPLIB website, problems *glass4* and *roll3000* could be solved by the state-of-the-art MIP solvers.

September 2006: The optimal values of *a1c1s1*, *timtab2*, and *swath* were found by GAMS Grid computing in Condor (Ferris 2006).

November 2006: The optimal values of *atlanta-ip*, *msc98-ip*, and *rd-rplusc-21* were found using Xpress 2006B by Vazacopoulos et al. (2006).

In this paper, we address several optimization strategies using Xpress-MP[1] that lead to finding the optimal solutions presented by Vazacopoulos et al. (2006) and strategies for finding the optimal solution to the problems *protfold* and *sp97ar*, previously unsolved (see §5). We also present improved solve times (to optimality) for five of the problems that have been solved in the past year (see §4), and present improved solutions for all the remaining six open problems (see §7).

Before considering the harder MIPLIB problems, we first look at some of the easier ones. The Xpress-MP performance on the group of MIPLIB problems that can be solved within one hour is investigated in §3.

3. Problems That Can Be Solved Within One Hour

Table 1 includes the 29 instances from MIPLIB 2003 for which Xpress 2006B can find the optimal solution within one hour using default settings. All tests were run on a dual Xeon 3.0 GHz, 64 bit, 4 GB of RAM, running Windows XP with Xpress being run in serial mode.

By default, Xpress will stop when the best-found MIP solution is within 0.01% of optimality. For our experiments, we carried on the tree search until optimality had been proven.

The most interesting points from the results of Table 1 are as follows:

- six problems can be solved at the root node;
- nine problems are solved by Xpress 2006B in less than one second;
- the Xpress 2006B average computing time on the group of easier problems is 164.6 seconds. If one disregards those problems solved within one second, then the average solve time of the remaining 20 problems is 265 seconds; and
- the three more challenging instances on the group of easier problems are *mas74*, *mzzv11*, and *pk1*, respectively, having Xpress 2006B solve times of 3,332, 359, and 228 seconds.

3.1. Historical Overview of Xpress Performance

We now look at how the Xpress performance has improved over the past few years and describe some of the reasons for these improvements.

Figure 1 shows the number of MIPLIB 2003 problems that could be solved by the last four annual releases of Xpress-MP within 30 minutes. Using the same computer (a dual Xeon 3.0 GHz, 64 bit, running XP), Xpress 2003G could only solve 22 instances, whereas Xpress 2006B now solves six more instances within a 30-minute time window.

Table 2 provides the speedup factor of the later Xpress releases (2004D, 2005B, and 2006B) with respect to Xpress 2003G for the easier group of MIPLIB 2003 problems. Instances (*fixnet6*, *gesa2*, *gesa2-o*, *modglob*, *modglob*, *p2756*, and *pp08a*) that could be solved by all the four solvers within one second have been disregarded.

The increase in performance of Xpress has not been down to a single enhancement but is due to numerous algorithmic improvements that combine to form a toolbox of techniques. If any one of these techniques is applicable to a particular model, then it can dramatically reduce the solve time of that model. Some of the algorithmic improvements are described below.

3.1.1. Presolve. Most of the MIPLIB problems benefit from presolving. Presolving techniques have been around for a while (see Brearley et al. 1975) and consist of techniques that are not too expensive in terms of computational time, but reduce the problem size and strengthen the MIP formulation. The benefits of reduced problem size are gained when solving each node of the tree search and usually give linear

[1] Version 17.10.04 of release 2006B has been used in these computational experiments.

Table 1 Branch-and-Bound (B&B) Nodes and Solution Times of the MIPLIB 2003 Instances, Which Are Solved to Optimality by Xpress-MP 2006B with Default Settings in One Hour

Problem	Xpress B&B nodes to		Xpress time† to		Optimal value
	Solution	Optimality	Solution (s)	Optimality (s)	
10teams	26	51	2	2	924
aflow30a	3,416	7,023	23	43	1,158
air04	166	185	36	36	56,137
air05	209	261	31	32	26,374
cap6000	1,389	1,937	7	9	−2,451,377
disctom	1	1	4	4	−5,000
fiber	56	69	<1	<1	405,935.18
fixnet6	9	11	<1	<1	3,983
gesa2	1	1	<1	<1	25,779,856.4
gesa2-o	1	1	<1	<1	25,779,856.4
harp2	30,942	67,239	90	149	−73,899,798.84
manna81	1	1	<1	<1	−13,164
mas74	46,709	2,380,685	35	3,332	11,801.1857
mas76	1	219,797	<1	159	40,005.0541
misc07	2,220	21,359	8	65	2,810
mod011	1,001	1,277	64	72	−54,558,535
modglob	1	11	<1	<1	20,740,508.1
mzzv11	1,047	1,145	345	359	−21,718
mzzv42z	121	125	48	48	−20,540
nw04	28	161	15	17	16,862
opt1217	1	1	<1	<1	−16
p2756	8	15	<1	<1	3,124
pk1	157,892	232,391	149	228	11
pp08a	124	213	<1	1	7,350
pp08aCUTS	88	181	<1	1	7,350
qiu	6,245	10,383	120	147	−132.873137
rout	4,165	10,323	34	67	1,077.56
set1ch	1	1	<1	<1	54,537.75
vpm2	19	1,341	<1	2	13.75

†Obtained on a dual Xeon 3.0 GHz, 64 bit, 4 GB of RAM, running Windows XP.

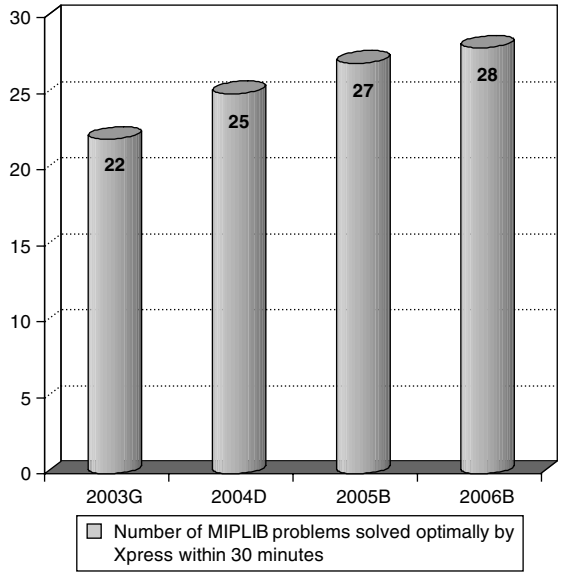

Figure 1 Number of MIPLIB 2003 Problems Solved Optimally by Xpress Within 30 Minutes Using Default Settings (on a Pentium 4, 3.6 GHz, Running Windows XP) Across Different Releases in the Past Four Years

speedups. However, strengthening the formulation using techniques such as coefficient tightening (see Savelsbergh 1994) can lead to reductions in the tree size and so can result in much larger speedups. The number of presolving techniques that are now available within Xpress has increased as we have analysed model formulations and found ways to strengthen them. Some of the formulations that were previously considered to be "bad" formulations can no longer be considered as such because presolve will essentially reformulate the problem.

3.1.2. Cutting Planes. The generation of cutting planes can have a huge impact on solution times. For some problems, branching has little effect on the LP solution and so the tree size grows exponentially. This is a particular problem when the LPs are degenerate and large numbers of nodes with the same objective value can be created. The *10teams*, *manna81*, and *opt1217* models from MIPLIB 2003 are typical examples. If good cuts are generated for these instances, the integrality gap can be closed, and it is then a matter of finding the optimal MIP solution from among the remaining degenerate LP solutions.

Table 2 Comparative Performance of the Xpress Software Releases Between 2003 and 2006 in the MIPLIB 2003 Easier Instances

Problem	Solve time† of 2003G (14.27) (s)	Optimality speedup factor to 2003G by 2004D (15.30.12)	2005B (16.10.09)	2006B (17.10.04)
10teams	451	30.1×	32.2×	**225.5**×
aflow30a	141	0.8×	0.8×	**3.3**×
air04	75	3.1×	**3.8**×	2.1×
air05	54	1.5×	1.2×	**1.7**×
cap6000	46	2.0×	2.2×	**4.6**×
disctom	≥20,000	≥**6,670**×	≥**6,670**×	≥5,000×
fiber	3	**3.0**×	**3.0**×	**3.0**×
harp2	437	1.4×	1.5×	**2.7**×
manna81	≥6,000	≥1,500×	≥**6,000**×	≥**6,000**×
mas74	3,781	1.2×	**1.3**×	1.1×
mas76	112	**1.1**×	0.9×	0.7×
misc07	98	1.1×	0.9×	**1.3**×
mod011	129	1.4×	1.4×	**1.6**×
mzzv11	97,889	<1.5×	**119.7**×	113.2×
mzzv42z	≥20,000	n/a	≥**415**×	≥385×
nw04	52	**4.7**×	4.0×	2.7×
opt1217	≥35,000	n/a	n/a	≥**35,000**×
pk1	**139**	0.6×	0.7×	0.6×
pp08aCUTS	2	**2.0**×	**2.0**×	**2.0**×
qiu	196	**1.3**×	**1.3**×	1.2×
rout	86	0.3×	0.9×	**1.2**×
set1ch	2,965	0.8×	**2,965**×	**2,965**×
vpm2	6	2.0×	2.0×	**3.0**×

Note. The fastest solution time is shown in bold.

†Obtained on a dual Xeon 3.0 GHz, 64 bit, 4 GB of RAM, running Windows XP.

The cuts generated within Xpress can be categorized as follows:

- Gomory/lift-and-project cuts;
- clique cuts;
- lifted cover cuts;
- mixed-integer rounding (MIR) cuts;
- implication cuts; and
- flow path cuts.

Gomory cuts (Gomory 1960) are general-purpose cuts that are generated from an optimal simplex tableau using the integrality of the fractional integer or binary variables. Lift-and-project cuts (Balas et al. 1993, Balas and Perregaard 2002, Cornuéjols 2007, Perregaard 2003) are disjunctive cuts that can be shown to be equivalent to Gomory cuts generated from a simplex tableau that is not necessarily feasible or optimal. These cuts can be generated for most of the MIPLIB problems and are usually very good at helping to close the integrality gap.

The remaining cuts work on various structures in the matrix or rely on finding certain properties. Clique cuts (see Atamturk et al. 1998) are cuts that restrict the sum of a set of binary variables to be less than one. Violated clique cuts can be generated from other cliques in the model. They are typically generated on airline crew-scheduling models such as the *air04*, *air05*, and *nw04* models in MIPLIB 2003.

Lifted cover cuts (Crowder et al. 1983, Van Roy and Wolsey 1987) are generated from knapsack constraints within the matrix from which a violated cover cut can be generated. The cover cut is then lifted to include other variables in the knapsack but not already in the cover. Xpress generates various cuts that are extensions to lifted cover cuts such as lifted generalized upper-bound (GUB) cover cuts, which are lifted cover cuts generated from knapsack constraints where the variables in the knapsack are members of clique inequalities (or GUBs). The *cap6000* model is a typical instance for which lifted GUB cover cuts can be generated.

MIR cuts (see Wolsey 1998) are essentially Gomory cuts generated from constraints or simple aggregations of constraints of the original problem. Like lifted cover cuts, MIR cuts can only be generated from constraints whose coefficients are nonuniform. Many of the MIPLIB 2003 problems contain knapsack-type constraints or constraints that can be aggregated to produce knapsack constraints.

Implication cuts (Hoffman and Padberg 1991) can be generated when a binary variable implies a bound on another variable. As an example, variable upper bounds (VUBs) can be generated as cuts when the original model contains aggregated VUB constraints. These cuts are very effective due to their sparsity.

Flow path cuts (Padberg et al. 1985) can be generated if a model has has a network structure for which nonzero flows incur a fixed charge. Flow path cuts can be very effective; for example, the reason why the *set1ch* model now solves in less than a second is due to the addition of flow path cuts.

Cordier et al. (1999) give a more detailed description of some of the cuts within Xpress.

3.1.3. Branching Variable Selection. Branching variable selection can make a large difference to the size of a tree search. It is usually best to select branching variables that have a big impact on the problem in terms of objective function change or structural change. The reduction in tree size from improving the branching variables selection can be sufficiently large to make it cost effective to spend time finding better branching variables. Strong branching (Applegate et al. 1995) is a look-ahead method that performs dual iterations on potential branching candidates to establish the effect of branching on the candidates. The method can be expensive, so the amount of strong branching is limited by default. The airline scheduling problems *air04*, *air05*, and *nw04* are typical instances that benefit from strong branching.

3.1.4. Node Preprocessing. The size of the tree search can be reduced by performing preprocessing at the tree nodes before solving them. This consists

of reduced cost fixing and bound tightening. In the MIPLIB 2003 test set, the *cap6000* model is one of the models to benefit from node preprocessing. For this model, the tree search can dive to a depth of 2,000 or more before the node becomes fathomed, but most of these branches can be avoided by node preprocessing.

3.1.5. Heuristics. For some problems, finding MIP solutions in the tree search is difficult. For example, the MIPLIB 2003 problem *disctom* is very degenerate, and branching on most of the fractional variables results in subproblems that have a similar number of fractional variables and exactly the same objective function value. The tree search dives very deep before detecting infeasibility, and so finding an integer solution is very difficult. However, it is fairly easy for rounding heuristics to find the optimal solution, and because there is no optimality gap at the top node, the problem solves in a few seconds.

3.2. Strategies for Solving Harder Problems

For the easier MIPLIB problems, we have seen that the techniques described above can be very effective. For the harder MIPLIB problems, preventing the exponential growth of the tree search is much more of a challenge. For example, running Xpress for a month would not be of much use if the size of the tree search keeps growing. As an example of the problem we face, consider a tree search where it is necessary to dive to a depth of 20 before nodes are fathomed. The tree size will be of the order of a million nodes, and if each node takes a second to solve, the problem will eventually solve in just under two weeks. Now, consider the case where the depth at which nodes are fathomed has increased to 40. The tree size will now be of the order of 1e+12 nodes, and even if the nodes now take 1/100 second to solve, the solution time will be around 350 years. Thus, the size of the tree search can easily become unmanageable, and when attempting to solve a problem it soon becomes apparent whether it has any chance of being solved.

What options do we have when trying to solve what appears to be an intractable problem? The first thing to consider is the formulation. Poor formulation can make a model much harder to solve, and it is not always possible for presolve to automatically improve the formulation. Formulating MIP models can be somewhat of an art, and it is difficult to give general advice on how to improve a formulation.

It is worth considering whether a model can be decomposed into smaller problems that are easier to solve. Sometimes a model contains a set of major decision variables that once fixed, the model splits into several smaller models. Each of these smaller models may be fairly easy to solve, but the combined model is much harder to solve. A strategy for solving the overall model then is to enumerate all possible combinations of the major decision variables and solve the smaller subproblems.

If the formulation cannot be improved, then we have to try various strategies for helping the tree search. Finding a good integer solution early in the tree search often helps prune the tree search and can help to improve the branching. For extremely hard problems, it is worth trying to find a very good solution in an initial run and then using the cutoff value provided by the solution in another run that tries to prove optimality. The settings for the two runs may be completely different.

For the easier MIPLIB problems, we have seen the importance of cutting strategies. Finding the right cuts can dramatically reduce the tree size. However, adding too many cuts can clog up the matrix and slow down the node optimizations. The default cut strategy can be too conservative for hard problems, so it is worth trying to increase the number of cuts added to the problem.

If it looks like the tree search has a chance of completing, we can always resort to a brute-force approach. We can try to reduce the size of the tree search by increasing the strong branching effort or alter the order in which the tree nodes are searched. Running parallel Xpress can also help. The parallel code performs separate tree search dives in different threads in a way similar to the operation of the earlier distributed parallel code described in Laundy (1999) and gives linear speedups in many cases.

4. Problems That Have Been Solved Recently

We now describe the strategies that we used to solve to optimality five problems—*a1c1s1*, *arki001*, *glass4*, *roll3000*, and *swath*—which were the last five confirmed problems solved prior to Vazacopoulos et al. (2006). We remark that one of the MIPs (*a1c1s1*) is solved for the first time using a single computer and another (*swath*) is solved in its original form for the first time using a single computer.

We describe the strategies in terms of the changes we made to the Xpress parameters (see the Xpress-MP Optimizer Reference Manual (Dash Optimization 2005) for a full description of the parameters).

4.1. *a1c1s1*

Problem *a1c1s1* is a lot-sizing problem first looked at by Van Vyve and Pochet (2001). For this problem, a large part of the initial integrality gap can be reduced by cutting, but the final 10% of the gap is very hard to close either by adding extra cuts or by branching. For this reason, alternative strategies were required.

The approach we took to solving this problem consisted of decomposing it into a set of easier problems. First, a partial branch and bound was run with full

strong branching to create an initial set of 128 subproblems. For each subproblem that could not be solved within half an hour, we split it again into five subproblems and repeated this action until all subproblems were solved.

The test computer used was an Intel Core Duo 2 running at 3 GHz with 4 GB of RAM. In the experiment, we used parallel Xpress by setting *mipthreads* = 2. The full time needed to prove optimality was about 32 hours.

The total number of subproblems created during the application of our method was 230, of which 212 represent "final" subproblems, i.e., problems that do not satisfy the splitting rule. The time needed to create these subproblems was 3,800 seconds, and the time needed to solve them was approximately 110,000 seconds, of which 72,000 seconds correspond to the "final" subproblems. The total number of nodes solved by our procedure was approximately 2.1 million nodes.

To solve each subproblem, we used several Xpress controls to get better optimization performance. Namely, we used a node selection rule that selects the node with the best LP bound from *all* outstanding nodes (i.e., *nodeselection* = 2), and put more effort on Xpress cutting and on strong branching (*cutstrategy* = 3, *covercuts* = 50, *gomcuts* = 10, *cutfreq* = 2, *cutdepth* = 20, *treegomcuts* = 0, and *sbeffort* = 2). We also used a cutoff value of 11,534.

Ferris (2006) has also recently reported solving *a1c1s1*. For this achievement, he used the GAMS Grid computing in the Condor system, spending 3,452 hours of CPU time. Because our CPU time was about 58 hours (considering the two CPUs used), our approach using Xpress 2006B needs about 60 times less CPU consumption than the approach of Ferris (2006).

Problem *a1c1s1* has an optimal objective value of 11,503.444125.

4.2. *arki001*

The problem *arki001* is a relaxation of a nonlinear problem arising in the metallurgy industry. Although it is possible to find good solutions to this problem, it was not proven that the best solution found to this problem was indeed optimal until 2006 when Balas and Saxena (2005, 2008) solved the problem with the use of cuts from the split closure. The solve time reported in Balas and Saxena (2005) was about 65 hours, of which 54 hours were spent generating rank-1 split cuts and 11 hours were spent solving the strengthened MIP.

Achterberg et al. (2006b) solved this instance in under nine hours and Achterberg (2006), using SCIP with aggressive strong branching and conflict analysis resolution, solved the problem in 5.2 hours, computing about 1.8 million nodes.

Vazacopoulos et al. (2006) report solving *arki001* with Xpress 2006A in four hours. Xpress 2006B solves the problem in 3.9 hours on a dual Xeon 3.0 GHz, computing (using one thread) 2.85 million nodes. To achieve this goal, it is necessary to use aggressive cut generation, both at the root node (including lift-and-project cuts) and during branch and bound, and reduce the effort spent in strong branching (*cutstrategy* = 3, *covercuts* = 5, *gomcuts* = 10, *lnpbest* = 150, *cutfreq* = 1, *treegomcuts* = 4, *heurstrategy* = 0, *varselection* = 3, and *sbeffort* = 0.1).

Problem *arki001* has an optimal objective value of 7,580,813.046.

4.3. *glass4*

The problem *glass4* is a nesting problem first studied by Luzzi (2002). This particular instance is much easier for Xpress to solve if a cutoff value is used because this allows reduced cost fixing to be performed and allows all nodes with a bound greater than the cutoff to be fathomed. We therefore used a two-stage process to solve it in which the first stage was dedicated to finding a good solution and the second stage was dedicated to solving the problem with the cutoff value from the first stage.

One of the reasons that it is hard to find very good solutions for *glass4* is that the objective function is dominated by the cost coefficient of one variable that at 1e+6 is much larger than all the other cost coefficients which are one or two. This causes the tree search to spend too much time searching for solutions that are only slightly better than the last solution found. To avoid this, we set *mipaddcutoff*, which is added to the value of any MIP solution that is found to give the new cutoff value. As a result, any nodes that have a bound that is not better by at least *mipaddcutoff* than the last solution found will be disregarded. Because there is often a difference of around 1e+8 between solutions, we set *mipaddcutoff* to −9.98e+7. We also increased the likelihood of finding solutions by setting the backtrack strategy for the tree search so that the search backtracks to the node with the best estimated solution value (*backtrack* = 1) and by setting the branching choice so that both child nodes are solved when diving (*branchchoice* = 1). With these settings, 14 solutions were found in 90 seconds on a 3 GHz Intel Core 2 Duo with the last one found being a solution with value 1,200,012,600.

Using the cutoff value derived from the best-known solution (1,200,012,600), Xpress 2006B proves that this really is the optimal solution in nine minutes on a 3 GHz Intel Core 2 Duo, computing almost 500,000 nodes.

Problem *glass4* has an optimal objective value of 1,200,012,600.

4.4. *roll3000*

The problem *roll3000* is a rolling stock problem first studied by Kroon (2002).

Xpress 2006A belongs to the first group of solvers that could solve this instance to optimality (see Achterberg et al. 2006b, Vazacopoulos et al. 2006). Special settings were used that modified the node selection strategy to select the node with the best bound (i.e., *nodeselection* = 2) and put more emphasis on cutting (in particular, in generating "extra" Gomory cuts). Xpress 2006A needed about 2.8 million nodes and 15.5 hours to provide this certificate of optimality on a 3.0 GHz dual Xeon.

With further special settings, this time can be reduced to just 13 minutes. To do this, it is necessary to increase the effort spent doing strong branching and increase the generation of Gomory and lift-and-project cuts at the top node and in the tree. With the special settings (*gomcuts* = 20, *lnpbest* = 100, *lnpiterlimit* = 50, *cutfreq* = 1, *treecovercuts* = 2, *treegomcuts* = 2, *heurstrategy* = 3, *sbeffort* = 4), Xpress 2006B needs approximately 100,000 nodes to prove optimality for this problem.

Ferris (2006) states that *roll3000* was solved in about 50 hours of CPU time using GAMS Grid computing in Condor. Achterberg (2006), using SCIP with aggressive strong branching and moderate conflict analysis, solved this problem in 10.3 hours by computing 4.1 million nodes using a computer based on a 3.2 GHz Pentium 4.

Problem *roll3000* has an optimal objective value of 12,890.

4.5. *swath*

The problem *swath* is a mission planning model for synthetic aperture radar surveillance. The model optimizes the tours for aircrafts and is similar to the travelling salesman problem (TSP).

It is much easier to solve TSP problems if problem-specific cuts are added. Ferris (2006) recently solved *swath* after adding five rounds of subtour elimination cuts, resulting in 32 extra constraints. The "enlarged" problem was solved in less than 20 minutes using a single machine with a standard MIP solver.

However, this problem can also be solved without problem-specific cuts by using the same optimization procedure that was used to solve problem *a1c1s1* (see §4.1). Using both cores of an Intel Core Duo 2 computer, the elapsed time needed to solve the problem *swath* to optimality was about 66 hours.

The total number of subproblems created during the application of our method was 516, of which 422 represent final subproblems. The time needed to create these subproblems was 900 seconds, and the time needed to solve them was approximately 234,000 seconds of which 79,000 seconds correspond to the "final" subproblems. The total number of nodes solved by our procedure was about 60 million nodes.

In this case, we have again used several Xpress controls to allow us to get better optimization performance. We used a node selection rule that chooses the node with the best bound from *all* outstanding nodes (*nodeselection* = 2) and we put more effort on Xpress cutting (namely, on lift-and-project cuts) and on strong branching (*cutstrategy* = 3, *covercuts* = 50, *gomcuts* = 20, *cutfreq* = 5, *cutdepth* = 50, *treegomcuts* = 0 and *sbeffort* = 5, and *sbiterlimit* = 100). We also used a cutoff value of 468.

We note that after more than 36,000 hours of CPU time, the GAMS Grid facility on the Condor system (see Ferris 2006) was unable to prove optimality for the original (i.e., unchanged) *swath* problem.

Problem *swath* has an optimal objective value of 467.407491 (Ferris 2006).

5. Problems That Have Been Solved for the First Time

We now turn our attention to some of the harder problems in MIPLIB 2003 that were unsolved prior to September 2006.

Three MIP problems from MIPLIB 2003—*atlanta-ip*, *msc98-ip*, and *rd-rplusc-21*—were solved for the first time using Xpress 2006B by Vazacopoulos et al. (2006). In this section, we describe the various algorithmic approaches used in Vazacopoulos et al. (2006) to get certificates of optimality for the previous problems. In addition, the optimal objective values for two problems *protfold* and *sp97ar* are presented for the first time.

A summary of the results is displayed in Table 3, which includes the computing times and the corresponding optimal objectives.

The following subsections describe the individual approaches that were used to solve these problems.

5.1. *atlanta-ip*

Problem *atlanta-ip* is a min-cost network problem with side constraints. Although the integrality gap is quite small, it is very difficult to close it completely.

An initial analysis of the model showed that it contained a number constraints that appeared to be

Table 3 Objective Value and Computing Time Needed to Find the Optimal Solutions of Four (Previously Unsolved) MIPLIB 2003 Problems Using Xpress-MP 2006B

Problem	Computer system	Solve time	Optimal objective
atlanta-ip	3 GHz Intel Core 2 Duo	10 hours	90.00987861
msc98-ip	3 GHz Intel Core 2 Duo	1.5 hours	19,839,497.005874
sp97ar	3 GHz Intel Core 2 Duo	>1 month	660,705,645.5
protfold	Xeon 3.0 GHz, 2 CPU, 4 GB of RAM	9 days	−31
rd-rplusc-21	Xeon 3.0 GHz, 2 CPU, 4 GB of RAM	3.6 hours	165, 395.2753

cuts and looked to be redundant at all MIP solutions. Adding cuts to a model can sometimes make it much harder to solve, so we dropped all the constraints we suspected were cuts.

Next, we observed that the objective function had a two-level structure. One set of binary variables has integer costs greater than or equal to one and the remaining binary variables have costs in the range 10^{-6} to 10^{-4}. If we drop the smaller cost coefficients, we create an easier problem because we can use the fact that the greatest common denominator of the remaining cost coefficients is one to improve the cutoff and speed up the tree search. Solutions to the easier problem will be feasible for the original problem, and because the sum of the smaller cost coefficients is less than 0.5, we can use the optimal solution value to the easier problem to add an objective cut involving only those variables with the larger costs to the original problem. In fact, the easier problem can be solved to optimality in roughly four hours (on a 3 GHz Intel Core 2 Duo computer) and 95,000 nodes. The contribution to the objective function from the smaller cost coefficients in the solution that we found was 0.012, so the solution found in this way is within 0.012 of the optimal solution to the original problem.

The final stage is to solve the original problem (with cuts dropped) and with the objective cut added. This took 4.5 hours and 66,000 nodes with the use of heavy cutting in the tree and strong branching (*gomcuts* = 0, *cutfreq* = 1, *cutdepth* = 20, *treegomcuts* = 0 and *sbeffort* = 5, and *sbiterlimit* = 500). Feeding the final solution back into the original problem showed that the constraints that we suspected were cuts were indeed redundant at this solution.

Problem *atlanta-ip* has an optimal objective value of 90.0098786144 (Vazacopoulos et al. 2006).

5.2. *msc98-ip*

The problem *msc98-ip* is a min-cost network problem similar to *atlanta-ip* (see §5.1) but arising from the design of a nationwide communication network.

To find this optimal solution, we have taken the same approach as the one used to solve problem *atlanta-ip*. Problem *msc98-ip* is somewhat easier to solve than *atlanta-ip*.

Xpress 2006B needs about eight minutes of computing time (on a 3 GHz Intel Core 2 Duo computer) for the second stage, from which a near-optimal feasible solution to the problem is found.

For the final stage, Xpress 2006B needs about one hour of computing time on the same computer, in which it computed almost 4,000 nodes using (as before) heavy cutting in the tree and strong branching.

Problem *msc98-ip* has an optimal objective value of 19,839,497.005874 (Vazacopoulos et al. 2006).

5.3. *protfold*

The problem *protfold* is a protein-folding model. The degeneracy in the model makes it particularly difficult to solve to optimality, and although it is easy to find poor-quality solutions, it becomes increasingly difficult to find better-quality solutions.

Running the problem for a few hours showed that the number of active tree nodes does not grow too rapidly, so there seemed to be some hope in applying a brute force approach to solving this problem. We left parallel Xpress running (with an aggressive heuristic strategy by setting *heurstrategy* = 3) and we were able to prove optimality for *protfold* in about nine days of (elapsed) computing time.

The computer used in this experiment was a 2 CPU Xeon 3.0 GHz with 4 GB of RAM and running Windows XP. The number of threads that we used for parallel Xpress was four (by setting *mipthreads* = 4). There was no particular reason for using this many threads, but it was clear from our previous experience on this problem that four threads would cooperate well in increasing the lower bound faster than if the more natural choice of two threads had been selected instead.

The optimal solution was found by Xpress at node 133,930 of the search tree after almost eight hours of computing time. At this particular node, the bound was precisely −35. To close this gap completely, Xpress computed nearly 2.4 million additional nodes.

Problem *protfold* has an optimal objective value of −31.

5.4. *rd-rplusc-21*

The problem *rd-rplusc-21* is a linearized relaxation of a nonlinear model for a distillation column with external reactor. The problem has a large number of constraints compared with variables, and these constraints are activated and deactivated by branching on the binary variables. The key to solving this problem is to find good estimates for the variable branching selection. Xpress 2006B can solve this problem by turning off three standard procedures—namely, by turning off cut generation (*cutstrategy* = 0), heuristics (*heurstrategy* = 0), and strong branching (*sbiterlimit* = 0). In addition, the solver has to pick the most promising candidates for variable selection using a method based on probing with dual estimates (*sbestimate* = 3), and from the selected candidates choose the candidate having the largest sum of the pseudo-cost degradation in both branches (*varselection* = 2). With these settings, Xpress 2006B solves *rd-rplusc-21* in 3.6 hours (using a dual Xeon 3.0 GHz), computing about 335,000 nodes. We finally remark that the *rd-rplusc-21* model contains 99 major decision variables that appear in a GUB constraint, and if we enumerate all 99 possibilities, the subproblems that are created can all be solved in 14 minutes.

Problem *rd-rplusc-21* has an optimal objective value of 165,395.2753 (Vazacopoulos et al. 2006).

5.5. *sp97ar*

The problem *sp97ar* is a railway line planning model proposed by van Hoesel et al. (2001). When trying to solve this problem, it soon becomes apparent that a brute-force approach will not work. Even if a tree search strategy that improves the best bound is used, the rate at which the best bound increases soon becomes extremely slow. The LP relaxation bound is 6.5256e+8, and this bound can eventually be increased to 6.578e+8, but increasing it above 6.6e+8 looks to be impossible.

The key to solving this problem is to use a technique that we call branch-and-bound fixing. For most of the 14,101 binary variables, fixing the binary variable to one increases the LP bound after cutting quite substantially. In fact, several variables can be fixed to zero as the bound increases above the value of the best-known solution. Thus, the problem size can be reduced by fixing each variable to one in turn and performing a tree search for a limited number of nodes using the best-known solution as a cutoff. If this tree search completes without hitting the node limit and without finding an integer solution, the variable that had been fixed one is said to be branch-and-bound fixed to zero. Several variables only require a few nodes to be branch-and-bound fixed, but it becomes increasingly difficult to fix variables in this way and eventual tree searches of up to half a million nodes are required. However, after over a month of CPU time, around 90% of the variables can be fixed to zero and the problem becomes small enough to be solved. The solution of this final problem still requires around 24 hours of CPU time with heavy use of strong branching.

Problem *sp97ar* has an optimal objective value of 660,705,645.5.

6. First Feasible Solution of *stp3d*

The MIP *stp3d* is possibly the hardest problem in MIPLIB 2003. The problem is a Steiner tree problem for which the LP relaxation and some of the tree node reoptimizations can be difficult. The first feasible solution of problem *stp3d* has been found by Xpress using a maximum degradation strategy (i.e., by setting *varselection* = 4). This solution was found in about five hours using Xpress 2006B specially designed to apply fast node reoptimizations. The first known solution of problem *stp3d* has an objective value of 529.778190.

Using the next generation of local search heuristics of Xpress, the first solution of *stp3d* was quickly improved to a solution with objective value of 500.736 (see §7). Note that this solution defines a 3.3% relative gap.

So far, the best-known solution for *stp3d* is 500.736.

Table 4 Improved Objective Values of the MIPLIB 2003 Unsolved Problems

Problem	Old best-known obj. value (z^{**})	Xpress improved obj. value (z^{*})	Gain $\lvert 1-(z^{*})/(z^{**})\rvert$ (%)
stp3d	Unknown	500.736	n/a
ds	283.4425	116.59	58.9
momentum3	370,177.036	236,426.335	36.1
t1717	193,221	170,195	11.9
liu	1,172	1,102	6.0
dano3mip	691.2	687.733333	0.5

7. Reducing the Gap of All Open Problems

Using the next generation of Xpress heuristics, based on local search procedures built on top of Xpress 2006B, improved solutions were found for all of the remaining (seven) open problems from MIPLIB 2003. Table 4 provides these improvements along with the previously best-known objectives, to our best knowledge.

The best-known objectives of the open problems were mostly taken from the discussion list of the MIPLIB 2003 website (Achterberg et al. 2006b). Several recent papers (Achterberg and Berthold 2005; Balas and Saxena 2005, 2008; Fischetti and Lodi 2003; Hansen et al. 2006) referencing MIPLIB 2003 were also considered to define the best objectives.

From Table 4, it can be seen that the new local search heuristics were capable of considerably improving the best-known solutions. The most notable achievement occurred with the *ds* problem, which consists of a 58.9% gain over the old best-known solution.

Each one of these solutions was found in a few hours on a standard computer on an average case. The precise solution times are not available because the improved solutions were found in an incremental way, and because the parameters would usually be different when applied between two consecutive rounds of heuristics.

Table 5 lists the remaining unsolved problems in MIPLIB 2003 and for each instance presents the best upper bound (i.e., objective) and lower bound to the optimum.

It can be seen that instances *momentum3*, *liu*, and *ds* have the largest relative gaps, having, respectively, gaps of 149.3%, 85.6%, and 52.8%. These problems still offer a challenge for Xpress, and it seems unlikely that they will be solved without making use of information associated with the problem logic or further breakthroughs in MIP technology.

At the other end, instance *stp3d* has a relatively small gap (3.3%), but the rate at which the gap can be

Table 5 Best-Known Lower and Upper Bounds of the MIPLIB 2003 Unsolved Problems

Problem	Lower bound (z^+)	Upper bound ($z^\star$)	Gap $\lvert 1 - z^\star/z^+ \rvert$ (%)
stp3d	484.71817	500.736	3.3
dano3mip	578.05603	687.733333	19.0
t1717	136,538.4219	170,195	24.6
ds	76.32504272	116.5	52.8
liu	613	1,102	79.8
momentum3	94,824.16406	236,426.335	149.3

closed by branching or cutting is relatively slow so it may be a while before this problem is solved.

8. Conclusions

In this paper, we have shown that it is possible to solve some of the hardest problems in MIPLIB 2003 using current technology. In particular, the solution methodology and technology considered led us to achieve the following results: (i) five MIP problems (*atlanta-ip*, *msc98-ip*, *protfold*, *rd-rplusc-21*, and *sp97ar*) were solved for the first time; (ii) one MIP problem (*a1c1s1*) was solved for the first time using a single computer; (iii) the first feasible solution for the *stp3d* problem was found; and (iv) improved solutions were found for the remaining open problems from MIPLIB 2003.

Although there have been great advances in MIP technology, tuning and modeling still play an important role when solving hard optimization problems.

References

Achterberg, T. 2006. Conflict analysis. Presentation, SCIP Workshop at ZIB, October 10–11, Konrad-Zuse-Zentrum fur Informationstechnik Berlin, Berlin, http://scip.zib.de/download/slides/SCIP-conflictAnalysis.ppt.

Achterberg, T., T. Berthold. 2005. Improving the feasibility pump. ZIB-Report 05-42, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin-Dahlem, Germany.

Achterberg, T., T. Koch, A. Martin. 2006a. MIPLIB 2003. *Oper. Res. Lett.* **34** 1–12.

Achterberg, T., T. Koch, A. Martin. 2006b. Mixed Integer Problem Library (MIPLIB) 2003. http://miplib.zib.de/.

Applegate, D., E. Bixby, V. Chvatal, W. Cook. 1995. Finding cuts in the TSP. Technical Report 95-05, DIMACS, Rutgers University, Piscataway, NJ.

Ashford, R. 2007. Mixed integer programming: A historical perspective with Xpress-MP. *Ann. Oper. Res.* **149** 5–17.

Atamturk, A., G. L. Nemhauser, M. W. P. Savelsbergh. 1998. Conflict graphs in integer programming. Technical Report LEC 98-03, Georgia Institute of Technology, Atlanta.

Balas, E., M. Perregaard. 2002. Lift-and-project for mixed 0-1 programming: Recent progress. *Discrete Appl. Math.* **123** 129–154.

Balas, E., A. Saxena. 2005. Optimizing over the split closure. Research Report MSRR-674, Tepper School of Business, Carnegie Mellon University, Pittsburgh.

Balas, E., A. Saxena. 2008. Optimizing over the split closure. *Math. Programming Ser. A* **113** 219–240.

Balas, E., S. Ceria, G. Cornuéjols. 1993. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Math. Programming* **58** 295–324.

Brearley, A. L., G. Mitra, H. P. Williams. 1975. Analysis of mathematical programming problems prior to applying the simplex algorithm. *Math. Programming* **8**(1) 54–83.

Cordier, C., H. Marchand, R. Laundy, L. Wolsey. 1999. bc-opt: A branch-and-cut code for mixed integer programs. *Math. Programming* **86** 335–354.

Cornuéjols, G. 2007. Valid inequalities for mixed integer linear programs. *Math. Programming* **112** 3–44.

Crowder, H., E. L. Johnson, M. W. Padberg. 1983. Solving large-scale zero-one linear programming problems. *Oper. Res.* **31** 803–834.

Dash Optimization. 2005. Xpress-MP Optimizer Reference Manual. Dash Optimization Ltd., Nothants, UK.

Dash Optimization. 2008. Explore Xpress-MP. http://www.fairisaac.com/xpressmp.

Ferris, M. 2006. GAMS, Condor and the grid: Solving hard optimization problems in parallel. Presentation, September 22, Department of Industrial and Systems Engineering, Lehigh University, Bethlehem, PA.

Fischetti, M., A. Lodi. 2003. Local branching. *Math. Programming Ser. B* **98** 23–47.

Gomory, R. E. 1960. An algorithm for the mixed integer problem. Technical Report RM-2597, RAND Corporation, Santa Monica, CA.

Guéret, C., C. Prins, M. Sevaux. 2002. *Applications of Optimization with Xpress-MP*. Dash Optimization Ltd., Nothants, UK. [Translated by Susanne Heipcke.]

Hansen, P., N. Mladenović, D. Uroević. 2006. Variable neighborhood search and local branching. *Comput. Oper. Res.* **33** 3034–3045.

Hoffman, K. L., M. Padberg. 1991. Improving LP-representations of zero-one linear programs for branch-and-cut. *ORSA J. Comput.* **3** 121–134.

Kroon, L. 2002. Personal communication.

Laundy, R. 1999. Implementation of parallel branch-and-bound algorithms in Xpress-MP. T. Ciriani, S. Gliozzi, E. Johnson, R. Tadei, eds. *Operational Research in Industry*. Palgrave Macmillan, Basingstoke, Hampshire, UK, 25–41.

Luzzi, I. 2002. Exact and heuristic methods for nesting problems. Ph.D. thesis, University of Padova, Padova, Italy.

Padberg, M. W., T. J. Van Roy, L. A. Wolsey. 1985. Valid inequalities for fixed charge problems. *Oper. Res.* **33** 842–861.

Perregaard, M. 2003. A practical implementation of lift-and-project cuts: A computational exploration of lift-and-project cuts with Xpress-MP. Presentation, 18th International Symposium on Mathematical Programming (ISMP 2003) (Copenhagen), August 18–22, Mathematical Programming Society, Phildelphia, http://www.dashoptimization.com/home/downloads/pdf/ismp2003_MP.pdf.

Savelsbergh, M. W. P. 1994. Preprocessing and probing for mixed integer programming problems. *ORSA J. Comput.* **6** 445–454.

van Hoesel, S., J. W. Goessens, L. Kroon. 2001. A branch-and-cut approach to line planning problems. Working paper, Erasmus University Rotterdam, Rotterdam, The Netherlands.

Van Roy, T. J., L. A. Wolsey. 1987. Solving mixed integer programming problems using automatic reformulation. *Oper. Res.* **35** 45–57.

Van Vyve, M., Y. Pochet. 2001. A general heuristic for production planning problems. CORE Discussion Paper 56, Center for Operations Research and Econometrics, Université catholique de Louvain, Louvain-la-Neuve, Belgium.

Vazacopoulos, A. 2006a. State-of-the-art optimimation using Xpress-MP 2006A. Preconference workshop presentation, INFORMS Practice Meeting (Miami), April 30, INFORMS, Hanover, MD, http://www.dashoptimization.com/home/downloads/pdf/Informs_Practice_2006_workshop.pdf.

Vazacopoulos, A., R. Laundy, G. Tavares. 2006b. State-of-the-optimization using Xpress-MP v2006. Presentation, INFORMS Annual Meeting (Pittsburgh), November 6, INFORMS, Hanover, MD, http://miplib.zip.de/paper/vazacopoulos2006.pdf.

Wolsey, L. 1998. *Integer Programming*. John Wiley & Sons, New York.