Chemical Engineering Greetings to prof. Sauro Pierucci on occasion of his 65th Birthday, AIDAC, 151-160 (2011).

Computational advances in solving Mixed Integer Linear Programming problems

To Professor Sauro Pierucci for leadership in Process Systems Engineering

Ricardo M. Lima and Ignacio E. Grossmann

Department of Chemical Engineering, Carnegie Mellon University,
5000 Forbes Ave., Pittsburgh, PA 15213, United States

In this paper, we identify some of the computational advances that have been contributing to the efficient solution of mixed-integer linear programming (MILP) problems. Recent features added to MILP solvers at the algorithmic level and at the hardware level have been contributing to the increasingly efficient solution of more difficult and larger problems. Therefore, we will focus on the main advances in terms of hardware, and algorithms in one commercial solver to demonstrate the advantages and disadvantages of the recent features. Special attention is given to the utilization of multiple threads, parallelization modes, and the integration of heuristics with the branch and cut algorithm. Two problems are used to show the advantages of some of the new options. The results show that while some of the new features may help on the solution of difficult problems, they can also reduce the performance on relatively easy problems.

1. INTRODUCTION

A large number of optimization problems in Process Systems Engineering (PSE) can be described by Mixed-Integer Linear Programming (MILP) models. Examples include the optimization of production operations including planning and scheduling, optimization of supply chains involving logistics and distribution, multiple period optimization (Grossmann, 2005), and process synthesis using simplified models without nonlinearities (Biegler et al., 1999). Ultimately in Enterprise Wide Optimization (EWO) planning and scheduling are integrated in order to avoid suboptimal decisions. Furthermore, these are integrated with the supply chain which leads to very large-scale problems when multiple productions sites, distributions centers, markets and multiple products are considered. In addition, there are countless applications of MILP models in other domains, such as macro and micro economics, finance (Cornuejols G. and Tutunku R, 2007), energy planning and unit commitment problems (see Padhy, 2004 for a review), and biological systems (Gregory et al., 2000). In the process industries, real world problems usually lead to large-scale models, due to the size of the system under study, but also because of models that involve multiple periods. Furthermore, often new variables and equations are introduced to replace nonlinearities by piecewise approximations, or for performing exact linearizations (eg. product binary and continuous variables).

Generally MILP problems can be solved using Linear Programming (LP)-based Branch & Bound (B&B) solvers or with stochastic search based solvers. The advantage of the first approach is that it provides rigorous lower and upper bounds on the solution, which in turn provide information regarding the optimality of the solution. MILP problems may be represented by the following formulation:

$$Z(X) = \min\{cx + fy : (x,y) \in X\}$$

where

$$X = \{(x,y) \in \Re_+^n \times \{0,1\}^p : Ax + By \geq b\}$$

and c, f, b are vector of constants, and A and B are matrices of constants. B&B algorithms rely on the start of a solution of a linear relaxation of the form:

$$Z(X) = \min\{cx + fy : (x, y) \in X\}$$

where

$$X = \{(x, y) \in \Re_+^n \times [0,1]^p : Ax + By \geq b\}$$

to provide upper and lower bounds on the solution. Broadly speaking the algorithm starts with the solution of the initial relaxed LP problem at the root node, where afterwards an integral variable, x_i with the fractional value x_i^0 is chosen, and two additional subproblems are defined by adding the inequality $xi \leq \lfloor x_i^0 \rfloor$ to the LP problem at the root node and the other by adding $xi \geq \lceil x_i^0 \rceil$. This procedure is repeated for each sub-problem, until the upper bound defined by integral solutions is equal to the lower bound given by fractional solutions. During the search the upper and lower bounds are used to prune branches of the tree (for a detailed description see Wolsey (1998)). B&B algorithms, however, may not be able to effectively solve large problems due to the exponential number of subproblems that may have to be solved, particularly when the LP relaxation is poor. In reality MILP solvers have implemented more sophisticated versions denoted by Branch & Cut (B&C) algorithms. In these algorithms, valid inequalities denoted by cutting planes are added to the linear relaxations in order to reduce the size of the feasible space defined by X without eliminating any feasible integer solution.

As mentioned before, real world problems tend to be large and exhibit an exponential complexity with the problem size (NP-hard). However, it is clear that in the recent years truly remarkable progress has been made in the solution of MILP problems. As an example, consider the classical Kondili MILP model (Kondili et al.1993) with 72 binary variables, 179 continuous variables and 250 constraints. In 1987 using Kondili's B&B, it took 908 seconds and 1466 nodes in a VAX-8600 to solve, while Shah et al. (1993) took 119 seconds and 419 nodes using a SUN Sparc. Finally, in 2003 using CPLEX 7.5 with a laptop IBM T-40 only 0.45 seconds was required (Mendez et al., 2006). A more standard benchmark for MILP problems, the MIPLIB 2003 (Achterberg et al, 2006), shows that in the beginning of 2003 there were more than 30 problems with unknown optimal solution, while in the beginning of 2011 only four problems remain to be solved (http://miplib.zib.de, accessed on Feb. 11, 2011).

The ability to solve more complex problems has been supported by: a) advances of computational resources in terms of speed and memory, leading to faster calculations, and with enough memory storage to tackle large amounts of data (ten years ago a Digital workstation with one 600MHz CPU was a faster computer, while nowadays a common laptop has 4 threads each one at 2.5GHz); b) developments of new and improved algorithms and pre-processing techniques; and c) modeling systems that speed up the definition of problems. In addition, decomposition strategies such as rolling horizon algorithms and spatial and temporal Lagrangean decompositions have helped to facilitate the solution of larger MILP problems.

The specific objective of this paper is to evaluate some of the recent advances of the MILP solver CPLEX. Although there are other effective solvers like XPRESS and Gurobi, we focus on CPLEX as it has a number of new unique features. In fact, CPLEX is currently a software package that has available a set of different solvers, tools, and interfaces for different languages and software. CPLEX started as a linear programming solver, and currently it can handle MILP and Quadratic Programs (QP) ranging from Mixed Integer Quadratic Programming (MIQP) to Mixed Integer Quadratic Constrained Programming (MIQCP). During the last 20 years CPLEX has gone through different types of developments. The current computational performance is the result of a combination of improvements on several areas, which can be broadly divided in the following five major areas:

1. LP solvers
    - Pre-processing
    - Algebra for sparse systems
    - Methods: primal, dual, barrier
    - Techniques to avoid degeneracy and numerical difficulties
2. Cutting planes

- Several cutting planes available

3. MILP pre-processing techniques
4. Heuristics
   - Node heuristics
   - RINS
   - Polishing
5. Parallelization
   - Search in the B&B tree
   - Barrier method
6. B&B
   - Dynamic search

LP solvers improvements. The performance of the LP solver is of paramount importance in the solution of MILP problems, since the B&C may have to solve a large number of LP subproblems in the enumeration tree. Currently, commercial software have available the primal, dual and network simplex, and the primal-dual log barrier algorithm. Bixby (2002) has made an extensive study about the performance of these solvers and reported an improvement of three orders of magnitude between 1992 and 2002 in the speed up solution of LP problems. These improvements are supported by a) pre-processing techniques to reduce the size of the initial problem; b) linear algebra advances to handle sparse systems of equations; c) advances in the primal and dual simplex; and d) the effective parallelization of barrier algorithms. An additional feature to help on the solution of LP problems, denoted by concurrent optimization is present in CPLEX. This feature allows the solution in parallel of LP problems using different solvers simultaneously. With this option the first thread will run the dual simplex, the second the barrier, the third the primal simplex and the remaining threads will be used by the barrier solver. Therefore, the barrier solver when applied to large problems may be faster than the simplex based solvers, since there is not an effective parallelized simplex available. However, due to the superior performance of simplex based solvers when they start from an advanced basis, these solvers are faster than the barrier solver when their performance is compared within a B&C framework. Table 1 shows the speed ups obtained when solving an LP problem (based on one model from Lima et al., 2011) with 60,390 equations 69,582 variables using CPLEX 7.1 with one thread and CPLEX 12.2 with eight threads in the same computer. The results show that the primal and dual simplex solvers in CPLEX 12.2 are 4.5 and 5.5 times faster than the respective solvers in CPLEX 7.1. Assuming that these solvers only use one thread, these improvements are due to improvements in the algorithms. In the other hand, the parallelized barrier algorithm in version 12.2 is only 5.4 times faster.

Table 1 Elapsed CPU time to solve the same LP problem using the solvers available in CPLEX 7.1 and 12.2.

	CPU time (s)	
Solver	CPLEX 7.1	CPLEX 12.2
Primal Simplex	205	45
Dual Simplex	281	51
Network Simplex	174	91
Barrier	97	18

Cutting planes. All commercial solvers have the capability of generating cutting planes as a function of the type of constraints present in the problem to solve. The goal of adding cutting planes is to improve the linear relaxations by eliminating regions of the feasible space without cutting off integer solutions. The solvers have their own technology to identify specific constraints in the problem and to add the appropriate cuts. For example if CPLEX detects characteristics of a multi-commodity flow network with arc capacities it will add multi-

commodity flow cuts. For more general MILP models it will generate Gomory cuts, which have shown to be very effective. Using standard options, the cuts are generated and added in an incremental way until the improvement of the relaxation is within a specified tolerance. Using CPLEX as an example to track the implementation of different cutting planes, CPLEX 6.0 (1998) had available a limited number of cuts to be generated, while CPLEX 8.0 (2002) could generate 9 types of cutting planes, and currently, CPLEX12.2 can generate 12 different types of cutting planes. Bixby and Rothberg (2007) have studied the impact of different features on the CPLEX performance and have found Gomory cuts to be the most effective. They have used CPLEX 8.0 to solve a set of 106 problems and studied the impact of deactivating some features. These authors state that deactivating all the types of cutting planes available, the mean performance of the solver decreased 53.7 times. The progress made in the implementation of cutting planes, may suggest that the user might be less concerned about the problem formulation in terms of defining a tight relaxation. However, from our experience, with the current implementation of cutting planes, a tighter formulation used with an old version of CPLEX may have more impact on the performance of the solver than the last version of CPLEX with a bad formulation.

Pre-processing and probing techniques. The pre-processing for MILP has two objectives 1) reduce the size of the problem; and 2) improve the formulation by tightening the linear relaxation without increasing the size of the problem. The pre-processing uses four different techniques: 1) identification of feasibility; 2) identification of redundancy; 3) improvement of bounds; and 4) rounding. The probing techniques are based on fixing binary variables to either 0 or 1, and then check logical implications to evaluate the possibility of fixing variables and improve coefficients. Pre-processing and probing are both formalized in Savelsbergh (1994) and Wolsey (1998).

Heuristics. Currently the MILP solvers may apply several heuristics in order to diversify the search by diving to lower parts of the tree where solutions are expected to be. Broadly speaking, CPLEX uses two types of heuristics: node heuristics and neighbourhood search heuristics. The node heuristics comprise different approaches involving deduction of values of some 0-1 binary variables that can be fixed given the value of 0-1 variables that were fixed previously, and fixing the values of 0-1 variables and solving LP subproblems. The set of node heuristics are applied initially at the root node and their impact is evaluated in order to define the most promising ones, which are then used during the branch and bound. In addition, if the user provides a starting solution that is not feasible, CPLEX has the option to apply specific heuristics to repair the infeasible solution. CPLEX 7.1 had only available node heuristics, while the current version has four types of neighbourhood search heuristics available: a) Local Branching (LB); b) Relaxation Induced Neighborhood Search (RINS) (Dana et al., 2005); c) Guided Dives (GD) (Dana et al., 2005); and d) evolutionary algorithms for polishing MIP solutions (Rothberg E. 2007). From these heuristics we will highlight two, the RINS and the solution polishing.

The RINS explores the neighborhood of the incumbent to find better solutions using an algorithm with two steps: 1) the binary variables with the same value in the incumbent and the linear relaxation are fixed; 2) a sub-MIP with the remaining binary variables free is solved. Obviously poor relaxations lead to large sub-MIP problems to solve. In terms of implementation the sub-MIP problems are not solved to optimality, and the heuristic is invoked with a given frequency. The solution polishing is based on the same idea of fixing some of the binary variables and then solving a sub-MIP problem. However, here the binary variables are fixed using genetic operators similar to the ones used in genetic algorithms. During the B&C a solution pool with integral solutions is kept, which are then used as seeds for the mutation and combination operators to generate a vector with some binary variables fixed. For each vector of binary variables, a sub-MIP is solved. If the solution of this sub-MIP is better than the incumbent, the incumbent is updated. The solution polishing requires at least one integral solution in order to be activated. Note that both neighborhood searches maintain the logic of the upper and lower bounds used in the B&C. The solution polishing heuristic focuses directly on the improvement of the incumbent by solving sub-MIP problems, which has the cost of slowing down the improvement of the best bound. Both heuristics and cutting planes contribute to prove optimality; the heuristics by improving the upper bound by finding integer solutions, while the cutting planes focus on improving the linear relaxation and therefore helping to move the best bound.

Parallelization. Currently, several MILP solvers offer the option to use multiple threads during the B&C search. CPLEX offers two modes a deterministic mode and an opportunistic mode. On the one hand the deterministic mode guarantees the invariance and repeatability of the search path and results, i.e. the path taken to the solution is always the same in different optimization runs of the same problem. On the other hand, in the opportunistic mode the path may be different each time we optimize the same problem since there is less synchronization during the search. This means that for problems where optimality cannot be proved within the maximum time set, the opportunistic mode may give different results on different optimization runs. In terms of CPU time performance, it is expected that using more threads should be faster, and that the opportunistic mode should outperform the deterministic mode. However, this may not be the case due to the non-deterministic nature of the branch & bound search.

B&B and additional tools. In addition to the B&C algorithm, CPLEX 12.2 has available a new algorithm to solve MILP problems called dynamic search. Currently, it is treated as a trade secret and details of the algorithm have not been disclosed. The only thing that is known is that it is based on the B&C algorithm.

CPLEX, and in general MILP solvers, have additional tools that may help to improve the performance of the MILP solvers, and help on the implementation of specific strategies to solve difficult MILP problems. These tools involve: a) a tuning option to automatically find the most efficient MILP solver options to minimize the CPU time; b) a solution pool built during the B&C. The goal of the tuning tool is to identify the MILP solver options that will improve the performance of the solver. This is motivated by the large number of options that MILP solvers offer, and their impact on the solution and performance of the solvers. In addition, the default options are set to work well for a large collection of problems. However, they may not be the most appropriate for a specific problem. The output result of the tuning tool is a new option file with a set of options that will improve the performance of the solver.

The goal of the solution pool is to create a pool of integral solutions during the B&C search, and provide multiple solutions at the end of the optimization. The solutions collected may be filtered through criteria such as solutions with a given percentage of the optimal solution, or diverse solutions according to a specific criterion. This allows the decision maker to inspect sub-optimal solutions, which can be justified whenever the model does not capture the full essence of the problem, when approximations are used, or when the data is not accurate.

2. EXAMPLES

The performance of the MILP solvers is usually evaluated for a set of problems with different characteristics in terms of size and types of constraints. However, in this work due to space limitations only two problems are studied. *Problem I* is an open problem from the GAMS library with the name poutil.gms. The objective of this problem is to minimize the total cost involved in a portfolio optimization for electric utilities (Rebennack et al. 2009). *Problem II* is a problem based on the MILP continuous time slot based model for scheduling of a multi-product single stage continuous processes proposed by Lima et al. (2011), where the goal is to determine the optimal schedule that maximizes an economical index. The size of these problems is given in Table 2.

Table 2 Size of the problems

	Problem I	Problem II
Equations	2,178	16,886
Variables	1,260	12,156
0-1 variables	773	5,938

In this work we will focus in some of the options that are not so common to tune, such as: a) the number of threads used; b) the parallel modes: deterministic and opportunistic; c) the solution polishing option; d) the use

of heuristics; and e) the tuning tool. For both problems, when the solution polishing option is used, it is activated after 60 seconds. In addition, for the second problem a two stage optimization is considered. The main idea behind it is the following: first make a quick search for an integral solution using the polishing option, and afterwards use the B&C to prove optimality. Whereas, the second stage optimization uses the solution of the first stage as a starting point, the tuning tool is only used for the first problem to tune the CPLEX options.

These problems are implemented in GAMS 23.5 (Brooke, Kendrick, Meeraus, & Raman, 1998) and solved on a machine running Linux with 8 threads Intel Xeon, 2.66GHz and 8GB of RAM.

3. RESULTS

Problem I can be solved to optimality in 950 seconds using one thread with CPLEX 12.2 using the default options (if solved with XPRESS 20.00, Gurobi 3.0.1, the times are 476 seconds, 486 seconds, while no solution was found using BDMLP 1.3 and XA 15.07a in 1000 seconds). Using CPLEX 7.1 with a time limit of 1000 seconds, the final solution is 279,070.5 (relative gap of 7.3%). These results show a clear improvement between CPLEX 7.1 and CPLEX 12.2, and weaker performance of CPLEX 12.2 when compared with other solvers using the default options. A careful analysis of the output log of the solvers CPLEX, XPRESS, and Gurobi shows that the current versions spend more time at the root node, generating cuts and trying heuristics, than CPLEX 7.1. For example, CPLEX 7.1 does not find any integral solution at the root node and adds the following cuts: one implied bound cut and 28 Gomory fractional cuts; while at the end of the root node CPLEX 12.2 has a solution with a relative gap of 9.1%, finds 6 integral solutions, and improves the LP relaxation by adding 12 implied bound cuts, 5 flow cuts, 28 mixed integer rounding cuts, 56 zero-half cuts, and 7 Gomory fractional cuts. Similar numbers are obtained with Gurobi 3.0.1 at the end of the root node: 10 integral solutions, and a total of 103 cuts.

Table 3 presents the results obtained solving this problem using different options in terms of the number of threads, the opportunistic and the deterministic mode, and with the solution polishing activated. The results clearly show that increasing the number of threads used reduces the CPU time required to reach optimality. When using eight threads with the opportunistic mode instead of a single thread the elapsed CPU time is reduced from 950 seconds to only 61 seconds. This shows a clear advantage of using multiple threads. The last row of the table displays the results obtained when the polishing option is activated after 60 seconds, which turns out to be a bad option. In this case CPLEX is able to find the optimal solution, but it is not able to prove optimality by improving the best bound.

Table 3 Results for Problem I obtained for different instances

			Objective function	
Instances	CPU time (s)	Gap (%)	RMIP	MIP
1	950	0.0	266,793.0	266,793.0
4D	211	0.0	266,793.0	266,793.0
4O	206	0.0	266,793.0	266,793.0
8D	95	0.0	266,793.0	266,793.0
8O	61	0.0	266,793.0	266,793.0
8D POL	1000	0.94	264,291.7	266,793.0

In order to evaluate if the performance of CPLEX can be further improved, the tuning tool is applied to Problem I. The option file generated by the tuning tool has the following options cutpass=-1, heurfreq=-1, probe=-1, varsel=4, which respectively set off the cutting planes generation, deactivates the heuristics, no probing, and sets the branching based on pseudo reduced costs. Basically, it turns off some of the algorithmic capabilities. The speed ups obtained with these options are presented in Table 4. Using the new options with one thread the CPU

time required is 67 seconds, instead of 950 seconds, while with eight threads it is only eight seconds, instead of 95 seconds. However, for this problem, the tuning tool requires 327 seconds to run. This extra time, however, might be justified if a similar problem is to be solved many times.

Table 4 CPU times obtained for Problem I using standard options and the options from the tuning tool.

	Standard options	Options from the tuning tool.
Threads	CPU time (s)	CPU time (s)
1	950	67
8D	95	8

Problem II is larger and more complex than Problem I, and optimality is not obtained within the time limit of 3600 seconds. First, in order to demonstrate the improvements of the current CPLEX version, Problem II was solved using CPLEX 7.1 and CPLEX 12.2. The older version was not able to find one integral solution in 3600 seconds, while the latter was able to achieve a solution with a relative optimality criterion of 3.4%, see Table 5.

Table 5 Performance of CPLEX 7.1 and 12.2 solving Problem II. In CPLEX 12.2 only one thread is used.

			Objective function	
Solver	CPU time (s)	Gap (%)	RMIP	MIP
CPLEX 7.1	3600	-	2,687.9	-
CPLEX 12.2	3600	3.4	2,669.0	2,580.5

Several instances were also studied for this problem. The best solution is obtained using eight threads with the deterministic parallel mode with the solution polishing activated after 60 seconds. All instances terminate with a relative optimality criterion less or equal than 3.5%. Figure 1 shows the objective function versus the node number for the different instances for this problem. The first solutions found by the solver are negative but, it is able to evolve towards positive values.

The results in Figure 1 show that with the heuristics deactivated, the performance of the solver is significantly reduced, when compared with the other cases where heuristics are used. Analyzing the impact of the parallelization modes, the opportunistic mode is able to find more integral solutions than the deterministic mode, see Figures 1 and 2. Regarding the number of threads used, increasing the number of threads with the opportunistic option does not lead to a considerable increase in the performance of the solver. However, increasing the number of threads from 1 to 4 or 8 in the deterministic mode has an impact on the number of solutions and the node where the integral solutions are found, see Figure 2. In addition, in Table 6 it is shown that using eight threads with the opportunistic mode leads to a better integral solution and a lower optimality gap than using eight threads with the deterministic mode. Figures 1, 2, and 3 show also that the number of solutions found in the tree, and the tree level where they are found is different on each instance. Using eight threads with the polishing option activated after 60 seconds, it finds solutions early in the search and it is reported to require fewer nodes. However, with the polishing option it is solving sub-MIP problems that do not count as nodes, but consume time. Therefore, the number of nodes reported is a misleading metric when heuristics are used. A better metric is the CPU time required to achieve a solution within a given optimality criterion as shown in Figure 3, where it is clear that the polishing option can find solutions considerably earlier than in the instance where it is not used. The two stage optimization strategy obtained the lower optimality gap, due to the best bound found. In this table is also shown that when the solution polishing is used for a large percentage of the time, the solver

spends more time solving sub-MIP problems than improving the best bound, see the two rows at the bottom of Table 6.

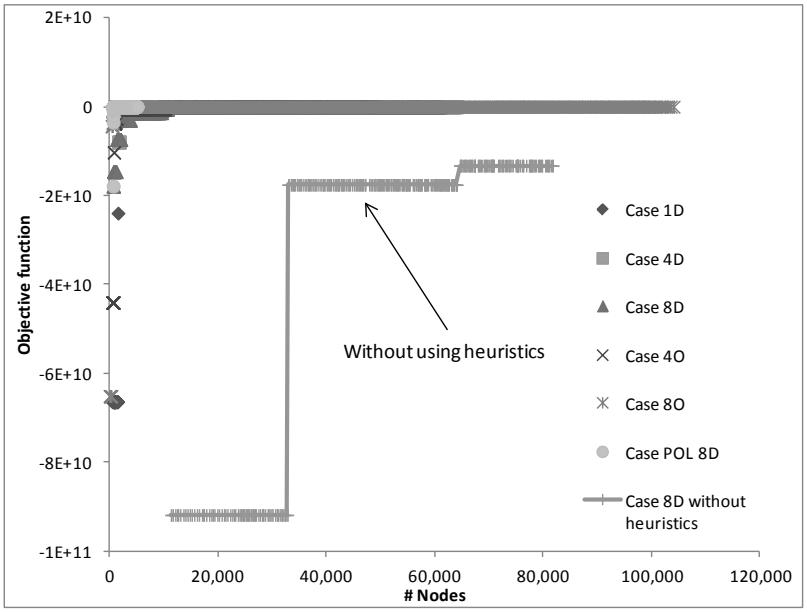

Figure 1 Objective function vs node number for different instances for Problem II. 1D, 4D, 8D – one, four and eight threads with the deterministic mode. 4O, 8O - four and eight threads with the opportunistic mode. 8D POL– polishing option activated after 60 seconds with 8D.

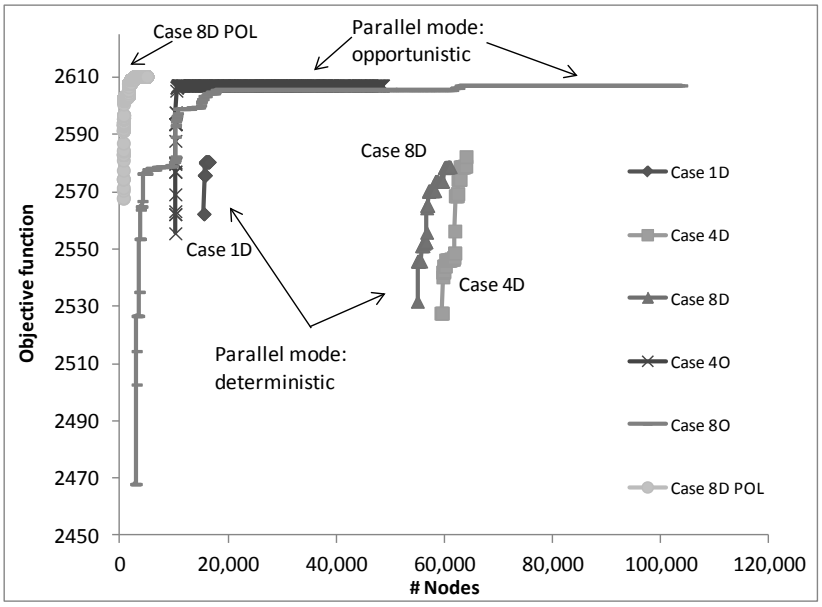

Figure 2 Objective function vs node number for different instances for Problem II. 1D, 4D, 8D – one, four and eight threads with the deterministic mode. 4O, 8O - four and eight threads with the opportunistic mode. 8D POL– polishing option activated after 60 seconds with 8D.

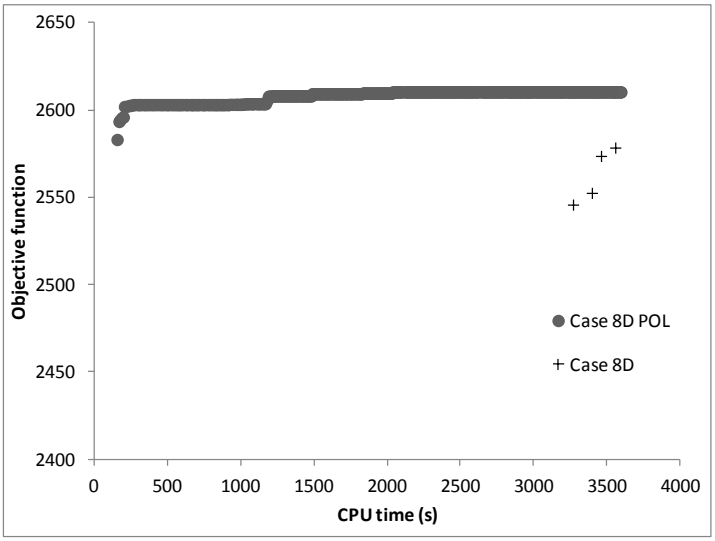

Figure 3 Objective function vs CPU time for different instances for Problem II, but considering only solutions with objective function greater than 2400. 8D - eight threads deterministic, 8D POL – polishing option activated after 60 seconds with 8D.

Table 6 Results for Problem II for different instances with different CPLEX options

			Objective function	
Instances	CPU time (s)	Gap (%)	RMIP	MIP
8D	3600	3.4	2,666.3	2,578.8
8O	3600	2.3	2,665.9	2,607.2
8D POL	3600	2.2	2,668.8	2,610.4
2 Stages optimization	3600	2.0	2,656.6	2,603.5

The introduction of heuristics in the B&C algorithm that use a random seed, for example the solution polishing, and the opportunistic parallel mode introduce a variability in the search path, which do not guarantee the repeatability of the results. This means that CPLEX may have a different performance in terms of computational time, final solution, and iterations, when the same problem is solved twice. This may be particularly important when the optimal solution is not obtained within the time limit set. In a development phase if repeatability is required to test different formulations, the above options should be avoided. However, they represent an opportunity to obtain better solutions.

4. CONCLUSIONS

With the current state of the art of MILP solvers, it is possible to solve problems that have been considered difficult or impossible to solve in the past. This has been possible by advances in the algorithms used, hardware, and modeling systems. In this work, instead of focusing on a large set of test problems, we focus in just two problems. The goal is to show the advantages of different options available in CPLEX that may work well in some problems, but in fact can also reduce the performance of the solvers on relatively easy problems. As an example, the deactivation of the heuristics in Problem I reduces by two orders of magnitude the computational

time, while in Problem II the heuristics help to find good solutions in the beginning of the search. An important aspect that was not discussed is the utilization of decomposition strategies, such as rolling horizon algorithms or Lagrangean decompositions. These have also contributed to address larger and more complex MILP problems, which together with the advances in MILP solvers represent an opportunity to address models that are larger in size and with increasing levels of detail.

5. REFERENCES

Atamurk, A., Nemhauser, G., Savelsbergh, M.W.P., 2000. Conflict graphs in solving integer programming problems. European Journal of Operational Research, 121, p. 40-55.

Achterberg T., Koch T., Martin A., 2006, MIPLIB 2003. Operations Research letters. 34(4), 361-372.

Biegler L.T., Grossmann I.E., Westerberg A.W., 1999, Systematic methods of chemical process design. Prentice Hall, New Jersey.

Bixby R, Rothberg E, 2007, Progress in computational mixed integer programming- a look back from the other side of the tipping point, Annals of Operation Research, 49(1), 37-41.

Bixby R.E., 2002, Solving real-world linear programs: a decade and more of progress. Operations Research. 50(1), 3-15.1

Bixby R.E., Fenelon M., Gu Z., Rothberg E., Wunderling R., 2000, MIP: Theory and practice – closing the gap, System Modeling and optimization methods, theory and applications, Kluwer Academic Publishers.

Cornuejols G., Tutunku R, 2007, Optimization methods in finance. Cambridge University Press.

Danna E., Rothberg, E., Le Pape, C., 2005. Exploring relaxation induced neighborhoods to improve MIP solutions, Mathematical Programming, 102(1), p. 71-91.

Fischetti M., Lodi, A., 2005. Local branching. Mathematical Programming, 98, p. 23-47.

GAMS corporation, CPLEX 2010 Manual.

Gregory L.M., Maranas C.D., Gutshall K.R., Brenchley J.E., 2000, Modeling and optimization of DNA recombination. Computers and Chemical Engineering. 24(2-7), 693-699.

Grossmann I., 2005, Enterprise-wide optimization: A new frontier in process systems engineering. AIChE Journal, 51(7), 1846-1857.

Kondili E., Pantelides C.C., Sargent W. H., 1993, A general algorithm for short-term scheduling of bath operations – I. MILP formulation. Computers and Chemical Engineering, 2, 211-227.

Land A. H., Doig, A. G., 1960, An automatic method for solving discrete programming problems, Econometrica, 28, pp 497-520

Lima R.M., Grossmann I.E., Jiao Y., 2011, Long-term scheduling of a single-stage multi-product continuous process to manufacture high performance glass, Computers and Chemical Engineering, (in press).

Linderoth J., 2004. Preprocessing and Probing for integer programs, DIMACS Reconnect Conference on MIP.

Mendez C.A., Cerda J., Grossmann I.E., Harjunkoski I., Fahl M., 2006, State-of-the-art review of optimization methods for short-term scheduling of batch processes, Computers & Chemical Engineering, 30 (6-7), 913-946.

Padhy N., 2004. Unit commitment - A bibliographical survey. IEEE T. Power Syst. 19 (2), 1196–1205.

Rebennack S, Kallrath, J, and Pardalos, P M, Energy Portfolio Optimization for Electric Utilities: Case Study for Germany. In Kallrath, J, Pardalos, P M, Rebennack, S, and Scheidt, M, Eds, Optimization in the Energy Industry. Volume 2. Springer, 2009.

Rothberg E., 2007. An evolutionary algorithm for polishing Mixed Integer Programming Solutions. INFORMS Journal On Computing, 19(4) p. 534-541.

Savelsbergh M.W.P., 1994. Preprocessing and probing techniques for Mixed Integer Programming problems. ORSA Journal on Computing, 6(4), p. 445-454.

Shah, N., Pantelides, C. C., & Sargent, W. H. (1993). A general algorithm for short-term scheduling of batch operations-II. Computational issues. Computers and Chemical Engineering, 2, 229 - 244.

Wolsey L.A., 1998, Integer programming. Wiley-Interscience, New York.