On the Optimization of CPLEX Models

Mohamad Kordafahar¹, Negar Jaberi², Reza Rafeh³

1. Malayer Branch, Islamic Azad University, Malayer, Iran
2. Department of Computer Science, Dolatabad Branch, Islamic Azad University, Isfahan, Iran
3. Arak University, Arak, Iran Department of Computer Engineering

*Corresponding Author email: mohammad.kordafshari@gmail.com

ABSTRACT: The performance of linear solvers is very dependent on their parameters and finding an optimal setting for parameters is a challenge. In this paper we propose a setting for CPLEX parameters to reduce the solving time of linear models. Our experimental results show that our setting is more efficient than the solver default settings.

Key words: CPLEX, Linear programming, Solver parameters, Combinatorial optimization problems.

INTRODUCTION

Combinatorial optimization problems are very important for industry because by solving them, companies maximize their benefit and minimize their cost. The main issue is that solving such problems is generally very hard because these problems are NP-Hard. In other words, there is no general and efficient algorithm for solving them. This is because the search space grows exponentially with the number of variable of the problem. The problems, such as, scheduling, planning, routing, investment, DNA sequencing, performance, timetabling and configure, are of the combinatorial optimization problems (Rafeh, 2008).

The combinational optimization problems can be divided into three categories:

Constraint Satisfaction Problems (CSP): A CSP is a problem composed of variables and constraints. For this kind of problems, it is adequate to find a solution which satisfies all constraints, and no solution has priority over another. Graph coloring, frequency assignment, n-queens and perfect squares are examples of constraint satisfaction problems.

Optimization Problems (OP): For optimization problems, finding a solution that satisfies all constraints is not enough. Instead, we need a solution which satisfies all constraints and minimizes (for minimization problems) or maximizes (for maximization problems) the objective function.

Over-Constrained Problems: In such problems, the number of constraints is so high that it is impossible to satisfy all of them (Tsang, 1993; Bartak, 1999). In this case the constraints are divided into two categories: Hard constraints: These constraints must be satisfied.

Soft constraints: These constraints must be satisfied as far as possible. However, some of them may be violated.

In over-constrained problems, the solutions must satisfy all hard constraints and the maximum possible number of soft constraints.

The most usual solving techniques for combinational optimization problems are divided into three categories:

Constraint Programming techniques (CP): These techniques consist of two consecutive steps: search and pruning the search space (Tsang, 1993).

Mathematical Methods (MM): These techniques are usually efficient and effective especially when the model is linear. The most well-known mathematical methods are simplex, interior point and branch and bound (Rafeh, 2008).

Local Search techniques (LS): These techniques avoid a complete search. Therefore, they may be useful for problems that their solving time with other techniques is very high. Since the search space is not explored
completely, LS techniques do not guarantee finding the solution and if the solution is achieved, it is unclear how far the solution diverges from the optimal one.

**MATHEMATICAL METHODS**

Mathematical methods are divided into two broad categories as follows.

**Linear Programming (LP)**
Such methods are applied to linear models which include real variables. The most well-known solving technique for linear models is Simplex (Chen et al., 2010). There are three simplex-based methods:
- The simple method for upper bounded variables
- The dual simplex method
- The revised simplex method

**Integer Programming (IP) and Mixed Integer Programming (MIP)**
Integer programming deals with linear models with integer variables. For MIP, models are linear and may include either real or integer variables (Rafeh, 2008).

The simplex algorithm is the basis of solving IP and MIP models. First, a relaxed version of the problem (i.e., no integer constraint on variables) is solved by linear methods. The following relation is always true for problems with MAX objective function $Z$:

$$Z_{ILP} \leq Z_{LP}$$

If in the solution all integer variables have integer values the solution is acceptable. Otherwise, the Branch and Bound technique is used. In general, the Branch and Bound algorithm is a search procedure which divides the feasible region of the IP problem successively into smaller regions and checks the possible solutions in those regions.

The preprocessing techniques try to find a better formulation for the problem to make the solving process simpler. Such techniques are used for tightening variables (making the domain of variables smaller), fixing variables, identifying redundant constraints and finding infeasibility regions for integer programming.

**Modelling Tools**
The modelling tools are divided into four categories.

**Constraint Languages**
Such languages are general programming languages and support a variety of solving techniques. They may be a logical language such as ECLiPSe (Apt and Wallace, 2007), an object oriented language such as Comet (Van Hentenryck and Michel, 2005), a functional language such as Hope (Burstall et al., 1980) or an imperative language such as Kaleidoscope (Lopez et al., 1994). The main advantage of such languages is their flexibility in modelling. They provide a rich set of constraints and allow users to define their own constraints. However, they do not support high level modelling. Therefore, their users must have enough programming skills.

**Constraint Toolkits**
Constraint toolkits are libraries which provide constraint modelling and solving facilities for programming languages. CPLEX (IBM ILOG 2010) and Localizer++ (Michel and Van Hentenryck, 2001) are examples of such toolkits. They do not support high-level modelling. As a result, their users must be skilled programmers. Also, some limitations may be imposed by the host language. The advantage of constraint toolkits is that their users do not need to learn a new language. In addition, data structures that are supported by the host language can help modellers to develop their libraries for specific domains.

**Mathematical Modelling Languages**
Mathematical modelling languages such as OPL (IBM ILOG, 2005), GAMS\(^1\) and AMPL (IBM ILOG, 2006) support high-level modelling and provide a syntax close to mathematical expressions. As a result, using modelling languages is easy for non-programmers. They allow users to use high-level structures, such as records or tuples,

\(^1\) http://www.gams.com
mathematical notations, arrays and sets and define the required constraints and functions and apply them into the model. However, current modelling languages are solver-dependent and can allow modelers to try a specific solving method to their models. For example, OPL uses mathematical and constraint satisfaction techniques.

**Specification Languages**

Specification languages are very high level and support sets, relations and functions. However, due to the large gap between the conceptual model and the design model, they cannot be used effectively for combinational and optimization problems (Rafeh, 2008).

**MIP Solvers**

CPLEX (IBM ILOG, 2010), Gurobi\(^1\) and MINOS (Holmström, 2008) are examples of MIP solvers. These are constraint solving toolkits suitable for MIP models. In this paper, we focus on the CPLEX solver using AMPL modelling language and investigate the configuration of its parameters.

CPLEX uses famous commercial optimization tools for MIP solvers. This solver is now available on the CPLEX website\(^2\). Currently, more than 1300 companies and government organizations with researchers of more than 1000 universities use CPLEX. CPLEX is very dependent of parameters and end users must often deal with its parameters especially for integer programming problems (IBM ILOG, 2010). Therefore, configuring parameters of MIP solvers like CPLEX is very important for efficiently solving the problems. We use CPLEX 12.1 and describe its parameters and their tuning for some problems in the next section.

AMPL is a perfect and powerful mathematical modelling language for linear, non-linear and mixed integer programming problems. This modelling language supports a variety of solvers such as CPLEX, Gurobi and MINOS (Fourer et al., 2003). In this paper, we model MIP problems using the modelling language AMPL and solve them by CPLEX. The main aim of the paper is to show how tuning CPLEX parameters properly can make the solving time shorter (IBM ILOG, 2006).

**CPLEX Parameters**

The most important parameters of CPLEX are as follows:

aggregate \((\text{default 1})\)

When \(i\) is 1, CPLEX looks for a variable \(x\) which is defined in terms of other variables.

The two-variable constraints of the form \(x = y + b\);

The constraint of the form \(x = \sum y\).

Under certain conditions, \(x\) can be eliminated from above equations by replacing its right-hand-side. When aggregate = -1, CPLEX decides how many passes must be performed to distinguish such functional dependencies. By setting \(i\) to a positive value, we determine the number of passes that should be done. When aggregate = 0, functional dependencies are ignored in order to save time and memory.

predual \((\text{default 0})\)

By default, after pre-solving the problem, CPLEX decides which one of the primal or dual methods can be used to solve the problem faster. predual = 1 explicitly tells CPLEX that the dual problem must be solved and predual = -1 means that the primal problem must be solved.

prereduce \((\text{default 3})\)

This directive determines which one of the primal or dual reductions, or both of them must be performed during preprocessing. When prereduce = 3, CPLEX performs both. When prereduce = 0, any reduction is prevented. When prereduce = 1, it performs only primal reduction and when prereduce = 2, it performs only dual primal reduction. Default setting is often adequate. Performing one of the two kinds of reductions may be useful to detect the boundlessness or infeasibility of the problem.

boundstr \((\text{default -1})\)

This directive tightens the bounds of variables. As a result some variables may become fixed removed from the branch & cut algorithm. By default (i.e. boundstr = -1), CPLEX automatically decides when to perform bound strengthening. This action usually improves the performance, but the repetition of it sometimes takes a long time. In cases where the bound strengthening time is long compared to the run-time, we can disable it by setting the parameter to 0. This directive becomes active by setting the parameter to 1 (boundstr = 1).

coeffreduce= \((\text{default 2})\)

---

1 http://www.gurobi.com
2 http://www.iloy.com/products/cplex
Coefficient reduction during the pre-solve phase usually improves CPLEX performance for integer programming. It is done by speeding up the LP sub-problems needs for the branch-and-bound algorithm.

However, when the coefficient reduces, linear sub-problems also get tightened and sometimes become more difficult to be solved. Thus, if CPLEX solves an IP problem in a number of nodes, but linear sub-problems take considerable amounts of time at these nodes, solving time may be improved by setting this directive to 0 which disables it. In this case, the number of nodes may be increased, but saving amount of time per node may compensate the increased number of nodes. By default, CPLEX, whenever possible, reduces the coefficients. When coefficient reduces = 1, CPLEX reduces only the coefficients of the integer values.

cutpass (default 0)

This directive controls a number of passes that CPLEX performs while cutting plans for a MIP model. By default (i.e. custpass = 0), CPLEX automatically determines the number of passes. This setting is sufficient for most problems. Cutting all generations is stopped by setting this directive to -1. By setting it to a positive integer, we can specify a particular number of passes.

eachcutlim (default 2100000000)

This directive limits the number of cuts of each type that may be generated. It may be useful for the models that a large number of models of one type of cut are generated. In this case, the overall limit on cuts is obtained before generating all types of cuts. By default, no limit is imposed on any cuts of each type. Only a general limit on cuts is imposed by applying the cutsfactor directive.

fraccand (default 200)

This directive limits the number of candidate variables that CPLEX will examine when it is generating fractional cuts on a MIP model. For most purposes, the default value of 200 is satisfactory.

repeatpresolve (default -1)

This directive tells CPLEX when re-applies pre-solve with or without cuts, to a MIP model after it completes processing of the root. The default value (repeatpresolve=1) allows CPLEX to choose pre-solving with or without cut. When repeatpresolve=1, pre-solve performs without cuts. When repeatpresolve=2, pre-solve performs with cuts and when repeatpresolve=0, pre-solve is prohibited.

fpheur (default 0)

This directive tells CPLEX when accepts the feasibility pump heuristic and its effects. The default value allows CPLEX to select heuristic or not. Setting fpheur = -1 determines that the feasibility pump heuristic should not be applied. The value of 1 tells CPLEX that it searches all of the solutions and value of 2 allows CPLEX to search the solutions with the good value of the objective function. Cost of the feasibility pump heuristic is high, but sometimes it can find solutions on the problems that CPLEX is unable to find.

heuristicfreq (default 0)

This directive is used to determine the frequency of each node that CPLEX applies heuristic at them. It can also be applied to find solutions that are lost due to use of other settings. The default value (heuristicfreq = 0) informs CPLEX to use internal logic to decide when to accept the heuristic. When heuristicfreq = -1 it stops heuristic at all nodes. Set heuristicfreq to a positive integer determines the frequency of nodes with which CPLEX applies the heuristic.

mipalgorithm (default 0) mipcrossover (default 1)

These directives determine an algorithm or the combination of algorithms that CPLEX uses to solve linear sub-problems at each branch & cut node. The values of these directives are shown in Table 01.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>mipalgorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>0</td>
</tr>
<tr>
<td>Primal simplex</td>
<td>1</td>
</tr>
<tr>
<td>Dual simplex</td>
<td>2</td>
</tr>
<tr>
<td>Network simplex</td>
<td>3</td>
</tr>
<tr>
<td>Barrier</td>
<td>4</td>
</tr>
<tr>
<td>Sifting</td>
<td>5</td>
</tr>
</tbody>
</table>

The default strategy selects the algorithm by using an internal heuristic based on the type of sub-problem. CPLEX usually applies Dual Simplex method when the problem has linear constraints and applies Barrier method when problem has quadratic constraints.
By setting mipalgorithm=4 and mipcrossover=1, CPLEX uses the Barrier algorithm to solve sub-problems, and primal simplex for the crossover. In the cases, dual simplex may be faster. When the sub-problems are quadratically constrained programs, CPLEX does not perform a crossover, so this directive has no effect.

mipcuts (default 0) or
cliquercuts (default 0)
covercuts (default 0)
disjcuts (default 0)
flowcuts (default 0)
flowpathcuts (default 0)
fraccuts (default 0)
gubcuts (default 0)
impliedcuts (default 0)
mfcuts (default 0)
mircuts (default 0)
zerohalfcuts (default 0)

IP solving time can be often improved by producing new constraints (or cuts) based on polyhedral considerations. These additional constraints tighten the feasible region. They reduce the number of fractional variables that CPLEX needs to select as a branching variable. CPLEX can generate cuts based on different combinations of directives listed above.

By default, CPLEX decides when to generate cuts. Usually the default setting results the best performance. To disable a family of cuts, the corresponding directive is set to -1. To enable the generation of cuts, the corresponding directive is set to 1 and to enable the cut generation more actively and aggressively, the directive must be set to 2.

startalgorithm (default 0)
This directive specifies the algorithm that CPLEX will use to solve the initial LP relaxation. Table 2 depicts the possible values of this directive.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>startalgorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic</td>
<td>0</td>
</tr>
<tr>
<td>Primal simplex</td>
<td>1</td>
</tr>
<tr>
<td>Dual simplex</td>
<td>2</td>
</tr>
<tr>
<td>Network simplex</td>
<td>3</td>
</tr>
<tr>
<td>Barrier</td>
<td>4</td>
</tr>
<tr>
<td>Sifting</td>
<td>5</td>
</tr>
<tr>
<td>Concurrent</td>
<td>6</td>
</tr>
</tbody>
</table>

**integrality (default 1.0e-5)**
In optimal solving a sub-problem, if a variable has distance \( r \) with an integer number, it is considered to have an integral value. For some problems, increasing \( r \) caused to accept the solution faster. This parameter may be set to 0 to improve the power of MIP solutions.

**Setting the CPLEX Parameters**
Default values for CPLEX parameters usually performs well for solving problems, however, by tuning these parameters may reduce the solving time. To show this, we set these parameters optimally for a bunch of well known problems. The examined problems in this article are as follows.
Perfect Squares: In this problem, the goal is to put several small squares in a large square without overlapping.
Job shop: In this problem, the goal is to complete some works by some limited car in minimum time, so that the number of request for a type of car is no more than the available number of it at any moment.
N-queens: In this problem, the goal is to place \( N \) queens into an \( N \times N \) chessboard, so that none of the queens does not threaten others.
Knapsack: In this problem, the goal is to put the number of things with different values and weights in a knapsack with limited capacity, so that the total weights of selected items does not exceed the capacity of the knapsack and the total value of selected items is maximized.

Products: In this problem, the goal is to determine how much of each product should be produced inside the company and how much outside, while minimizing the overall production cost, meeting the demand, and not exceeding the resource constraints.

Table 3 depicts optimal values of the parameters for the above problems.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Optimal for Perfect Squares</th>
<th>Optimal for Job Shop</th>
<th>Optimal for N-queens</th>
<th>Optimal for Knapsack</th>
<th>Optimal for Products</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregate</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>predual</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>prereduce</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>boundstr</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>coeffreduce</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>cutpass</td>
<td>0</td>
<td>1</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>eachcutlim</td>
<td>21*10^-1</td>
<td>100</td>
<td>21*10^-1</td>
<td>21*10^-1</td>
<td>21*10^-1</td>
<td>21*10^-1</td>
</tr>
<tr>
<td>fraccand</td>
<td>200</td>
<td>12</td>
<td>15</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>repeatresolve</td>
<td>-1</td>
<td>-1</td>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>fpheur</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>heuristicfreq</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>micalgorithm</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mpicrossover</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>mpicuts</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>startalgorithm</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>integrality</td>
<td>1.0e-5</td>
<td>1.0e-1</td>
<td>1.0e-3</td>
<td>1.0e-5</td>
<td>1.0e-5</td>
<td>1.0e-5</td>
</tr>
</tbody>
</table>

Evaluation

In this section, we compare the execution times of the models using the default settings and optimal settings for CPLEX parameters. All models have been executed on a PC with 2.61 GHZ AMD Athlon 64 * 2 dual core Processor 5000 +, and 1 GB RAM, 1 MB cache running Windows XP. Table 4 depicts the experimental results.

<table>
<thead>
<tr>
<th>Model</th>
<th>Default Settings</th>
<th>Optimal Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perfect Squares</td>
<td>20.64</td>
<td>1.23</td>
</tr>
<tr>
<td>Job Shop</td>
<td>31.76</td>
<td>5.84</td>
</tr>
<tr>
<td>N-queens</td>
<td>18.24</td>
<td>18.24</td>
</tr>
<tr>
<td>Knapsack</td>
<td>1.21</td>
<td>1.21</td>
</tr>
<tr>
<td>Products</td>
<td>1.1</td>
<td>1.1</td>
</tr>
</tbody>
</table>

As shown in Table 4, default settings of parameters works well for three problems: N-queens, Knapsack and Products. However, tuning parameters makes the Perfect Squares and Job Shop 17 and 5.5 times faster, respectively.

1 Inside production: a company manufactures its products in its own factories and each product consumes a certain amount of each resource.
2 Outside production: a company unlimitedly buys productions from other companies.
CONCLUSION

CPLEX is one of the most powerful MIP solvers. Default settings of CPLEX parameters usually work well for most of the problems. However, setting the parameters may considerably decrease the execution time of some problems. In this paper, we achieved a significant improvement on solving time of two problems, Job Shop and Perfect Squares.

In the future, we intend to find a smart method to set these parameters optimally and automatically based on the model.

REFERENCES

Barták R. 1999. "Constraint Programming – What is behind?", Constraint Programming for Decision and Control,
Burstall RM, MacQueen DB, Sannella DT. 1980. "Hope: An Experimental Applicative Language". Conference Record of the 1980 LISP Conference, Stanford University,
Chen DS, Batson RG, Dang Y. 2010.Applied Integer Programming, WILEY,
Kenneth H, Edvall MM. 2008.USER'S GUIDE FOR TOMLAB /MINOS. 1260 SEBishop Blvd Ste E, Pullman, WA 99163, USA: Tomlab Optimization Inc,
Van Hentenryck P, Michel L. 2005.Constraint-Based Local Search: MIT Press,