On the Fast Creation of Conflict Graphs for Integer Programs: A Computational Study

Samuel Souza Brito and Haroldo Gambini Santos [1,2]

Computing Department
Universidade Federal de Ouro Preto - UFOP
Ouro Preto, Brazil

Abstract

Conflict graphs are employed in modern Integer Programming to store relationships between variables. This information is crucial for an effective pre-processing and cut generation. CGs can be dynamically built inside branch-and-bound algorithms while analyzing the impact of the last fixations performed, but the earlier the CG is populated the faster strong inequalities and implications can be generated. On this work we present techniques which allow the fast creation of densely populated CGs at the root node. The impact of the availability of these large GCs for cut generation is evaluated by the inclusion of these techniques in the state-of-the-art open source integer programming solver COIN-OR CBC and its use in a cutting plane algorithm.

Keywords: conflict graphs, integer programming, clique cuts, set packing polytope

[1] Email: `samuelsouza@iceb.ufop.br`
[2] Email: `haroldo@iceb.ufop.br`

1 Introduction

In this work we present an approach for the creation of conflict graphs for Integer Programming (IP) problems. A Conflict Graph (CG) represents logical relations between binary variables. This kind of graph has a vertex for each binary variable and its complement. An edge between two vertices indicates that variables involved cannot be set to some specific value at the same time without violating one or more constraints.

CGs are typically constructed using probing techniques [4] based on constraints analysis. The probing technique consists in analyzing logical implications generated by fixing binary variables. For example, in a given problem the activation of the variable x implies the deactivation (i.e. activation of its complement) of the variable y, in order to respect the constraints of the problem. In this case, we found a an implication in the form $x = 1 \Rightarrow y = 0$, which would add the edge (x, y) to the graph.

Building a graph by looking for pairwise of conflicts may be computationally prohibitive when the input problem is large. Thus, the computational efficiency of this technique depends on the complexity of the constraint exploration, causing a trade off between efficiency and effectivity.

One important application for CGs is the generation of cutting planes derived from the Set Packing Polytope[17], such as the Clique Inequalities. The dynamic inclusion of these inequalities allows tightening the linear relaxation of an IP problem, improving performance of branch-and-bound based solvers [2]. Hoffman and Padberg [14] used conflict graphs to generate valid inequalities for set partitioning problems arising in airline crew-scheduling. Achterberg [1] presented heuristics based on SAT techniques for Mixed Integer Programming solvers to generate valid inequalities from the current infeasible subproblem and the associated branching information. The same idea has been developed independently and in parallel by Sandholm and Shields [18].

On this work we propose techniques to speed up the creation of dense conflict graphs at the root node, so that large CGs can be available at the start of the search process for the generation of strong inequalities. For ease of understanding of our approach we consider only pure Binary Programs (PB). Despite this, it can be applied to any Integer Program containing binary variables.

The rest of the paper is organized as follows. In Section 2, we formally explain our approach to build conflict graphs as well our strategy to speedup the detection of logical implications. In Section 3, we present the clique cut separation routine, including a clique extension step. In Section 4, the computational experiments with MIPLIB 2010 instances [15] and instances of the International Nurse Rostering Competition [12] are presented and analyzed. Finally, in Section 5, we conclude and give future directions about this work.

2 Conflict Graphs in Integer Programming

A conflict graph represents logical relations between binary variables. For two binary variables, we may discover four possible logical relations, using the notation of [2]:

$$
\begin{aligned}
x = 1 \Rightarrow y = 1 &\iff & x + (1 - y) &\leq 1 \qquad (1)\\
x = 1 \Rightarrow y = 0 &\iff & x + y &\leq 1 \qquad (2)\\
x = 0 \Rightarrow y = 1 &\iff & (1 - x) + (1 - y) &\leq 1 \qquad (3)\\
x = 0 \Rightarrow y = 0 &\iff & (1 - x) + y &\leq 1 \qquad (4)
\end{aligned}
$$

Given an Integer Programming (IP), a conflict graph can be constructed using probing techniques based on feasibility considerations. The basic idea is to analyze the impact of activation or deactivation (i.e. activation of its complement) of two variables at time, for each pair of variables and each constraint. Each constraint $i \in \{1, \ldots, m\}$ can be written as:

$$\sum_{j \in N} a_{ij} x_j \leq b_i \tag{5}$$

where N is the index set of binary variables x, a_{ij} is the coefficient for variable x_j at constraint i and b_i is the right-hand side of constraint i. Suppose we are analyzing two particular variables $x_{\hat{j}}$ and $x_{\hat{k}}$ with respect to constraint i. Consider that these variables are assigned with values u and v, respectively. Let:

$$L_i^{x_{\hat{j}}=u,\, x_{\hat{k}}=v} = \sum_{j \in N_i^- \setminus \{\hat{j}, \hat{k}\}} a_{ij} + a_{i\hat{j}} u + a_{i\hat{k}} v \tag{6}$$

where $N_i^- = \{j \in N : a_{ij} < 0\}$. In this case, $L_i^{x_{\hat{j}}=u,\, x_{\hat{k}}=v}$ is a lower bound for the value on the left-hand side of the constraint i, considering the assignments $x_{\hat{j}} = u$ and $x_{\hat{k}} = v$. If $L_i^{x_{\hat{j}}=u,\, x_{\hat{k}}=v} > b_i$, there is a conflict between the assignments of $x_{\hat{j}}$ and $x_{\hat{k}}$.

Performing these steps for each combination of values of two binary variables, considering each pair of variables in each constraint, leads to the creation of a conflict graph for any IP problem in $O(m \times n^2)$. For problems with many variables and constraints this technique may be too expensive computationally (See Experiments section). Nevertheless, for some constraint types a large number of conflicts can be quickly discovered. This is the case of the Generalized Upper Bound constraints ($\sum_{j \in N} x_j \leq 1$). As discussed in [2], even handling explicitly conflict graphs induced by these constraints requires special data structures such that in the previous decade most solvers could not use all information which could be inferred just from GUB constraints. The following subsection will describe additional cases where cliques in constraints can be quickly detected (i.e., faster than $O(n^2)$). The following notation will be used: $\tilde{a}_{ik}$ is the k-th smallest coefficient in constraint i and $\acute{a}_{ik}$ indicates its index. Constants n_i and S_i^- denote the number of non-zero variables and the sum of all negative coefficients of constraint i, respectively. To illustrate one case, GUB constraints can be described as $\tilde{a}_{ik} = b_i = 1 \; \forall \; k$. In the next subsection fast clique detection will be discussed for additional constraint structures.

2.1 Fast detection of cliques in less structured constraints

We describe two simple cases where large cliques of conflicting variables can be detected just by traversing constraints with coefficients of variables sorted in non-decreasing order. Thus, conflicts in these constraints are discovered in $O(n \log n)$.

To detect cliques that involve the activation of some variables we start computing the lower bound of the left-hand side (LHS) for each two consecutive variables, starting the analysis from the smallest coefficient ($k = 1$) to the highest one ($k = n_i$). The first step is compute the sum of negative coefficients excluding the two analyzed variables:

$$D_i^{x_{\acute{a}_{ik}}, x_{\acute{a}_{ik+1}}} = S_i^- - min(0, \tilde{a}_{ik}) - min(0, \tilde{a}_{ik+1}) \tag{7}$$

Thus, the lower bound for the LHS of constraint i when variables with k and $k+1$ smallest coefficients are fixed at one is:

$$LHS_i^{x_{\acute{a}_{ik}}=1, x_{\acute{a}_{ik+1}}=1} = D_i^{x_{\acute{a}_{ik}}, x_{\acute{a}_{ik+1}}} + \tilde{a}_{ik} + \tilde{a}_{ik+1} \tag{8}$$

Since $LHS_i^{x_{\acute{a}_{ik}}=1, x_{\acute{a}_{ik+1}}=1}$ is monotonically non-decreasing as k increases, if $LHS_i^{x_{\acute{a}_{ik}}=1, x_{\acute{a}_{ik+1}}=1} > b_i$, then there is a clique involving the activation of all variables from position k until position n_i. Moreover, we can discard the existence of such cliques by checking if $LHS_i^{x_{\acute{a}_{in_i-1}}=1, x_{\acute{a}_{in_i}}=1} \leq b_i$.

The same idea can be used to detect cliques that involve the deactivation of some variables, computing the lower bound of the LHS for each two consecutive variables, starting from the highest coefficient ($k = n_i$) to the smallest one ($k = 1$). The lower bound for the LHS of constraint i when variables $x_{\acute{a}_{ik}}$ and $x_{\acute{a}_{ik-1}}$ are fixed at zero is equal to $D_i^{x_{\acute{a}_{ik}}, x_{\acute{a}_{ik-1}}}$, which is obtained by equation 7: $LHS_i^{x_{\acute{a}_{ik}}=0, x_{\acute{a}_{ik-1}}=0} = D_i^{x_{\acute{a}_{ik}}, x_{\acute{a}_{ik-1}}}$

In the same form, $LHS_i^{x_{\acute{a}_{ik}}=0, x_{\acute{a}_{ik-1}}=0}$ is monotonically non-decreasing as k decreases. If $LHS_i^{x_{\acute{a}_{ik}}=0, x_{\acute{a}_{ik-1}}=0} > b_i$, then there is a clique involving the deactivation of all variables from position k until the variable of first position ($k = 1$). Moreover, we can discard the existence of such cliques by checking if $LHS_i^{x_{\acute{a}_{i2}}=0, x_{\acute{a}_{i1}}=0} \leq b_i$.

When no clique is found we use the pairwise analysis. Nevertheless, in our experiments we observe a large number of constraints that have cliques, speeding up the creation of conflict graphs.

Example 2.1 Consider the following constraints as a part of an IP problem, where all variables are binary:

$$+x_1 + x_2 + x_3 \geq 2 \tag{9}$$

$$-2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 4 \tag{10}$$

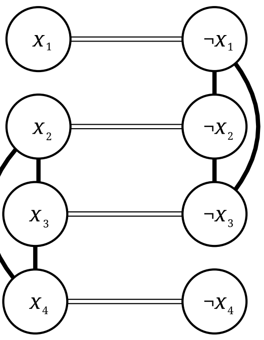

Fig. 1. Conflict graph for equations 9 and 10

Figure 1 shows the conflict graph for this constraints, where $\neg x_i$ represents the complement (or deactivation) of variable x_i. Edges represent conflicts between variables or complimentary variables (double edges). For the constraint of equation 9, we have to transform it in a $\leq$-constraint, multiplying by -1. So, we have: $-x_1 - x_2 - x_3 \leq -2$. How the coefficients of this constraints are ordered, we can start calculating $L_9^{x_{\hat{j}}=1, x_{\hat{j}+1}=1}$. For any pair of activated variables we are able to produce a solution that respects this constraint. For this reason we do not find a clique with active variables in this constraint. Calculating $L_9^{x_{\hat{j}}=0, x_{\hat{j}+1}=0}$ for the first pair of subsequent variables (i.e. $\hat{j} = 1$) we find a conflict that implies in a clique involving all complement of the variables starting from x_1 until x_3 ($L_9^{x_1=0, x_2=0} = -1$). So, we insert this conflicts in the graph.

We proceed analyzing the constraint of equation 10. For any pair of deactivated variables we are able to produce a solution that respects this constraint. For this reason we do not find a clique with deactivated variables in this constraint. Calculating $L_{10}^{x_{\hat{j}}=1,\, x_{\hat{j}+1}=1}$ for the consecutive pairs of variables we can found a clique starting from x_2 until x_4 ($L_{10}^{x_2=1,\, x_3=1} = 5$). We finish inserting this conflicts in the graph.

3 Cutting Planes

Linear programming relaxations can be significantly strengthened by the inclusion of inequalities derived from the set packing polytope (SPP) [17]. The most common classes of cuts for SPP are the clique cuts and the odd-hole cuts. A clique inequality for a set C of conflicting variables has the form $\sum_{j \in C} x_j \leq 1$ and an odd-hole inequality with conflicting variables C can be defined as: $\sum_{j \in C} x_j \leq \lfloor \frac{|C|}{2} \rfloor$. It is well known that in practice clique cuts are by far the most important ones [4]. The impact of these cuts has been explored for some hard timetabling problems [3,7]. Considering generic clique separation routines, the most common ones are the star clique and the row clique method [9,13,4]. These are fast separation routines which are used in the current version of the COIN-OR Cut Generation Library.

Our algorithm proposal considers aggressive clique separation: instead of searching for *the* most violated clique inequality we search for *all* violated clique inequalities. Some previous results indicate that this is the best strategy. In [7], for example, although authors used a branch-and-bound code to search for the most violated clique, computational results motivated the inclusion of non-optimally violated cuts found during the search. This result is consistent with reports of application of other cuts applied to different models, such as Chvàtal-Gomory cuts [10]. The option for inserting a large number of violated inequalities at once is also responsible for reviving the gomory cuts importance [8].

Our proposed clique separation routine has two main components:

(i) a module to separate all violated cliques in the conflict subgraph induced by the fractional variables;

(ii) a lifting module which extends generated cliques considering the original conflict graph.

The clique separation module was implemented using an improved version of

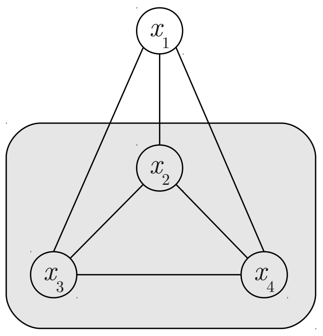

Fig. 2. Example of a k_3 which could be lifted to a k_4

the Bron-Kerbosch algorithm [6]. This version implements an optimized pivoting rule [5] to speed up the discovery of maximal cliques with large weight. This rule assigns the highest priority for visiting first nodes with large modified degree (summation of node degree and of its neighbors) and weight. Although this algorithm has an exponential worst case performance, the heuristic pivot rules make the algorithm suitable not only for running in the enumeration context but also for executing with restricted times, since larger violated cliques tend to be discovered first. Nevertheless, our experiments showed that all violated inequalities for all instances can be enumerated in a fraction of a second using our implementation. It is also important to remark that even if a subset of cliques is inserted, the optimal solution would not be missed, branching would take care of the rest. The importance of lifting clique inequalities can be explained with the conflict graph in Figure 2. Nodes inside the gray area indicate variables with non-zero values in the fractional solution. In this solution, only nodes $x_2, \ldots, x_4$ could contribute to define a maximally violated clique inequality. Nevertheless, subsequent linear programming relaxations could include three different violated k_3 [3] cliques by alternating the inactive variable. If the k_4 clique inequality were inserted during the separation of the first fractional solution, additional re-optimizations of the linear program could be saved. Furthermore, a less dense constraint matrix may be obtained with the insertion of these dominant constraints first.

It is well known that the separation of odd-holes contributes only marginally for lower bound improvement [4,16]. Nevertheless, its inclusion in the branch-and-cut procedure is cheap, since these inequalities can be separated in polynomial time using shortest path algorithms [11]. Odd hole inequalities can be strengthened by the inclusion of a wheel center, such as variable x_6 in the

[3] a clique with three nodes

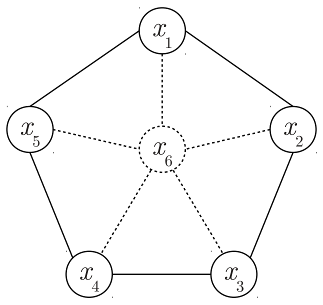

Fig. 3. Example of an odd hole and its possible extension to a wheel

conflict graph presented in Figure 3. In fact, for an odd hole with variables C and W being the set of candidates to be included as wheel centers of C, the following inequality is valid:

$$\sum_{j \in W} \lfloor \frac{|C|}{2} \rfloor x_j + \sum_{j \in C} x_j \leq \lfloor \frac{|C|}{2} \rfloor \tag{11}$$

4 Experimental Results

5 Conclusions and Future Work

References

[1] Achterberg, T., *Conflict analysis in mixed integer programming*, Discrete Optimization **4** (2007), pp. 4 – 20, mixed Integer Programming {IMA} Special Workshop on Mixed-Integer Programming.

[2] Atamtrk, A., G. L. Nemhauser and M. W. Savelsbergh, *Conflict graphs in solving integer programming problems*, European Journal of Operational Research **121** (2000), pp. 40 – 55.

[3] Avella, P. and I. Vasil'ev, *A Computational Study of a Cutting Plane Algorithm for University Course Timetabling*, Journal of Scheduling **8** (2005), pp. 497–514.

[4] Borndorfer, R., "Aspects of set packing, partitioning, and covering," Ph.D. thesis (1998).

[5] Brito, S. and H. G. Santos, *Pivoting in the Bron-Kerbosch algorithm for maximum-weight clique detection (in portuguese).*, 2011.

[6] Bron, C. and J. Kerbosch, *Algorithm 457: finding all cliques of an undirected graph*, Commun. ACM **16** (1973), pp. 575–577.
URL http://doi.acm.org/10.1145/362342.362367

[7] Burke, E., J. Mareek, A. Parkes and H. Rudov, *A branch-and-cut procedure fortheudine course timetabling problem*, Annals of Operations Research **194** (2012), pp. 71–87.

[8] Cornuéjols, G., *Revival of the Gomory cuts in the 1990\'s*, Annals of Operations Research **149** (2007), pp. 63–66.

[9] Eso, M., "Parallel branch and cut for set partitioning," Ph.D. thesis, Cornell (1999).

[10] Fischetti, M. and A. Lodi, *Optimizing over the first Chvàtal closure*, Mathematical Programming B **110** (2007), pp. 3–20.

[11] Grotschel, M., L. Lovasz and A. Schrijver, "Geometric Algorithms and Combinatorial Optimization," Springer, 1993.

[12] Haspeslagh, S., P. de Causmaecker, A. Schaerf and M. Stølevik, *The first international nurse rostering competition 2010*, Annals of Operations Research (2012), pp. 1–16.

[13] Hoffman, K. and M. Padberg, *Solving airline crew scheduling problems by branch-and-cut*, Management Science **39** (1993), pp. 657–682.

[14] Hoffman, K. L. and M. Padberg, *Improving LP-representations of zero-one linear programs for branch-and-cut*, ORSA Journal on Computing **3** (1991), pp. 121–134.

[15] Koch, T., T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy and K. Wolter, *MIPLIB 2010*, Mathematical Programming Computation **3** (2011), pp. 103–163.

[16] Méndez-Díaz, I. and P. Zabala, *A cutting plane algorithm for graph coloring*, Discrete Applied Mathematics **156** (2008), pp. 159–179.

[17] Padberg, M., *On the facial structure of set packing polyhedra*, Mathematical Programming **5** (1973), pp. 199–215.

[18] Sandholm, T. and R. Shields, *Nogood learning for mixed integer programming*, in: *Workshop on Hybrid Methods and Branching Rules in Combinatorial Optimization, Montréal*, 2006.