Creation of Conflict Graphs for Integer Programming: a Computational Study

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

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

Abstract

This is a short example to show the basics of using the ENDM style macro files. Ample examples of how files should look may be found among the published volumes of the series at the ENDM home page (`http://www.elsevier.com/locate/endm`)

Keywords: Please list keywords for your paper here, separated by commas.

1 Introduction

In this work we present an approach for the creation of conflict graphs for Integer Programming (IP) problems. A conflict graph 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 the variables involved cannot be set to some specific value at the same time without violating some constraint of the problem.

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

Conflict graphs are typically constructed using probing techniques [3] 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 logical relation of the form "$x = 1 \Rightarrow y = 0$". This logical relation can be represented in a conflict graph, creating an edge between the vertice associated with x and the vertice associated with y.

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. We also use probing techniques based on feasibility considerations, by analyzing constraint-by-constraint. However, we sort the coefficients and compute the bounds of the left-hand side for each constraint and compare with the respective right-hand side, in order to detect constraints that form cliques, avoiding the pairwise analysis of the variables in these constraints. The use of such strategy contributes to accelerate the conflict discovery process as shown in the performed experiments.

An important application for the use of conflict graphs is the clique cut generation, which performs a crucial role in the discovery of strong inequalities [4] in IP problems. The dynamic inclusion of these inequalities allows tightening the linear relaxation of an IP problem, crucial to the branch-and-bound based solvers [2]. Moreover, conflict graph can also be used to develop heuristics and branching primal schemes. Hoffman and Padberg [6] 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 [8]. With our experiments we can demonstrate that the generation and insertion of clique cuts contribute significantly to strengthening the linear relaxation.

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 1, we formally explain our approach to build conflict graphs as well our strategy to speed 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 [7] and instances of the International Nurse Rostering Competition [5] 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:

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

Given a Binary 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 BP 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:

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

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 $x_{\hat{k}}$. For example, if $u = 1$ and $v = 1$ there is a conflict

between the activation of these variables. Consequently, the logical relation $x_{\hat{j}} = 1 \;\Rightarrow\; x_{\hat{k}} = 0$ was found.

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. 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. The constants n_i and S_i^- will represent the number of 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 less structured constraints.

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. Thus, conflicts in these constraints are discovered in $O(n \log n)$.

To detect cliques that involve the activaction 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 coefficent ($k = 1$) 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 then if $LHS_i^{x_{\acute{a}_{ik}}=1, x_{\acute{a}_{ik+1}}=1} > b_i$, 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 $LHS_i^{x_{á_{in_i-1}}=1, x_{á_{in_i}}=1} \leq b_i$.

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 a IP problem, where all variables are binary:

$$x_1 + x_2 + x_3 \geq 2 \tag{9}$$
$$-2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 4 \tag{10}$$

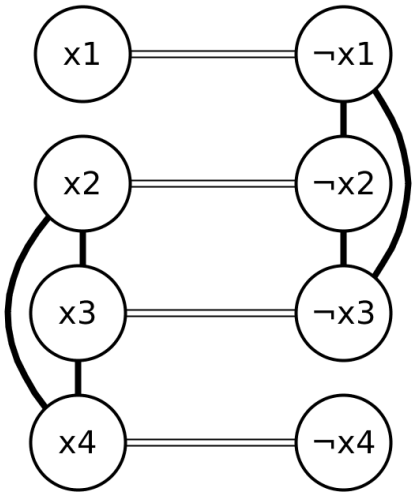

Fig. 1. A conflict graph for constraints 9 and 10.

Figure 1 shows the conflict graph for this constraints, where $\neg xi$ represents the complement (or deactivation) of variable xi. Initially, we create a vertex for each variable and its complement. Since in a feasible solution we can either activate a variable or its complement, we insert a edge between them (double lines). For constraint 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 constraint 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 Clique Cut Separation

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

[4] Chvátal, V., *Edmonds polytopes and a hierarchy of combinatorial problems*, Discrete mathematics **4** (1973), pp. 305–337.

[5] 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.

[6] 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.

[7] 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.

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