Creation of Conflict Graphs for Integer Programming: a Computational Study

Samuel Souza Brito[1]

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

Haroldo Gambini Santos[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 re-

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

lations 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, represented by the vertices, can not be activated (assigned with value 1) at the same time without violating some constraint of the problem.

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 compute the bounds of the left-hand side for each constraint and compare with the respective right-hand side, in order to avoid the analysis of constraints that would not lead to the discovery of conflicts. Furthermore, this comparison allows the detection of 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, especially on problems that have constraints on the type Generalized Upper Bound (GUB constraints).

For ease of understanding of our approach we consider only pure binary IP problems. Despite this, it can be applied to any IP problem 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 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. A vertex in this kind of graph represents a binary variable or its complement. An edge between two vertices means that the variables represented by these vertices can not be activated (assigned with value 1) at the same time without violating some constraint of the problem. For two binary variables, we may discover four possible logical relations:

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

Given a Binary Integer Programming (IP), a conflict graph can be constructed using probing techniques based on feasibility considerations. The basic idea is analyzing 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. It is noteworthy that the relation between a variable and its complement is stronger: exactly one of them must be assigned with 1 in any feasible solution. Each IP constraint i can be written as:

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

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:

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

where $N_i^- = \{j \in N : aij < 0\}$. In this case, $L_i^{x_{\hat{j}}=u,\, x_{\hat{k}}=v}$ is the minimum value of the left-hand side (i.e. lower bound) 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. However, for problems with many variables and constraints this technique can be computationally prohibitive.

By sorting the coefficients of a constraint in non-decreasing order we can found conflicts that forms a clique, avoiding making some pairwise analysis. These cliques are stored in a data structure used by the conflict graph. Let $x_{\hat{j}}$ and $x_{\hat{j}+1}$ two consecutive binary variables in constraint i. If $L_i^{x_{\hat{j}}=1,\, x_{\hat{j}+1}=1} > b_i$, there is a clique involving the activation of all variables starting from $x_{\hat{j}}$ until the last coefficient of i according to the non-decreasing order. Moreover, this approach can be used to avoid analyzing constraints that do not have such types of clique. Let k the index of the highest coefficient, namely the last one, in i. Thus, if $L_i^{x_{k-1}=1,\, x_k=1} \leq b_i$, there are no cliques involving the activation of variables in this constraint.

Similarly, we look for cliques between the complement of some variables by sorting the coefficients in non-increasing order. If $L_i^{x_{\hat{j}}=0,\, x_{\hat{j}+1}=0} > b_i$ there is a clique involving the complement of all variables starting from $x_{\hat{j}}$ until the last coefficient of i according to the non-increasing order. This can also used to avoid fruitless analysis. Let k the index of the smallest coefficient in i. Thus, if $L_i^{x_{k-1}=1,\, x_k=1} \leq b_i$, there are no cliques involving the complement of variables in this constraint.

When the absence of cliques can not be detected and no clique was 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

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.