Integrating Benders decomposition within Constraint Programming

Hadrien Cambazard, Narendra Jussien
`email:` {hcambaza,jussien}@emn.fr

École des Mines de Nantes, LINA CNRS FRE 2729
4 rue Alfred Kastler – BP 20722 - F-44307 Nantes Cedex 3, France

1 Introduction

Benders decomposition [1] is a solving strategy based on the separation of the variables of the problem. It is often introduced as a basis for models and techniques using the complementary strengths of constraint programming and optimization techniques. Hybridization schemes have appeared recently and provided interesting computational results [4, 5, 7, 8]. They have been extended [2, 3, 6] to take into account other kinds of sub-problems and not only the classical linear programming ones. However, decomposition has never been proposed to our knowledge in a generic constraint programming approach. This paper discusses the way a decomposition framework could be embedded in a constraint solver, taking advantage of structures for a non expert user. We explore the possibility of deriving logic Benders cuts using an explanation-based framework for CP and describe Benders decomposition as a nogood recording strategy. We propose a tool implemented at the top of an explained constraint solver that could offer such a systematic decomposition framework.

2 Context

Explanations for constraint programming. An explanation records information to justify a decision of the solver as a domain reduction or a contradiction. It is made of a set of constraints C' (a subset of the original constraints) and a set of decisions $dc_1, dc_2, \dots dc_n$ taken during search. An explanation of the removal of value a from variable v, $expl(v \neq a)$ will be written $C' \wedge dc_1 \wedge dc_2 \wedge \cdots \wedge dc_n \Rightarrow v \neq a$. An explanation is computed for any contradiction during the search and intelligent backtracking algorithms that question a relevant decision appearing in the conflict are then conceivable.

Principles of Benders decomposition. Benders decomposition can be seen as a form of *learning from mistakes*. It is based on a partition of the variables into two sets: x, y. The strategy can be applied to a problem of the form P :

$$\begin{array}{l|l|l}
\text{P : Min } f(y) + cx & \text{SP : Min } cx & \text{DSP : Max } u(a - g(\overline{y})) \\
\text{s.t : } g(y) + Ax \geq a & \text{s.t : } Ax \geq a - g(\overline{y}) & \text{s.t : } uA \leq c \\
\text{with : } y \in D, x \geq 0 & \text{with : } x \geq 0 & \text{with : } u \geq 0
\end{array}$$

A master problem considers only the y variables. A sub-problem (SP) tries to complete the assignment on x. If it is possible, the problem is solved, but if not, a cut (a constraint rejecting at least the current assignment on y) is produced and added to the master problem: it is called a Benders cut. This cut is the key point of the method, it has the form $z \geq h(y)$ (z represents the objective function – $z = f(y) + cx$) and is inferred by the dual of the sub-problem (DSP). So, even if the cut is derived from a particular $\overline{y}$, it is valid for all y and excludes a large class of assignments. From all of this, it can be noticed that duals variables or multipliers[1] need to be defined to apply the decomposition. However, a generalized scheme has been proposed in 1972 by Goeffrion [2]. Hooker [3] proposed also to enlarge the classical notion of *dual* by introducing an *inference dual* available for all kinds of sub-problems. They suggest a different way of thinking about duality: a Benders decomposition based on *logic*. For a discrete satisfaction problem, the resolution of the dual consists in computing the infeasibility proof and determining under what conditions the proof remains valid: this is exactly what explanations are designed for.

3 A decomposition approach in CP

In this paper, we consider problems which can be represented by P :

$$
\begin{array}{rl|rl|rl}
\text{P}: & \text{Min } obj & \text{MP}: & \text{Min } z & SP^k: & \text{Min } sz_k \\
 & \text{s.t}: Ct(x,y) & & \text{s.t}: Ct^i(x,y) & & \text{s.t}: Ct(x_k,\overline{y}) \\
 & \text{with}: x \in D_x, y \in D_y & & z < \overline{z} & & x_k \in D_x \\
 & & & y \in D_y & &
\end{array}
$$

$Ct(x, y)$ denotes a set of constraints on variables x, y and obj can be equal to $\{f(x, y), f(y), 0\}$. The problem P will be denoted $\{P_{xy}, P_y, P_0\}$ according to the corresponding objective functions. The decomposition scheme is done among x and y. We suppose that the remaining problem over x can be formulated using n sub-problems exhibiting strong intra-relationships and weak inter-relationships. Ideally, they should be as small and independent as possible to ensure the remaining sub-problem to be easy. So we make the assumption for the sub-problem to offer such an ideal (denoted by P) or approximate structure (denoted $P^{'}$, so we get in the same way $\{P^{'}_{xy}, P^{'}_{y}, P^{'}_{0}\}$). Master problem and sub-problems have then the generic form MP and SP (where $Ct^i(x, y)$ is the union of $Ct(x, y)$ and the benders cut gathered at iteration i).

3.1 Benders cuts as explanations

The Benders cut is a logic expression over the y variables, generated from the sub-problem solution. The cut must ensure that the algorithm terminates and finds the optimal solution. At iteration k, the added Benders cut must have the following properties:

[1] Referring to linear programming duality.

1. It is valid; it does not exclude any feasible solution over the x, y variables of the original problem (according to the current upper bound of z).
2. It must exclude at least the current instantiation $\overline{y}$ of the master that has been proved as sub-optimal or inconsistent

(2) ensures the termination of the algorithm and (1) ensures optimality as the master problem is proved to remain a valid relaxation and to provide a lower bound of P. As the explanation is a subset of the decisions taken by the master, it excludes at least the current assignment. An empty set indicates an infeasible P whereas the complete set excludes only $\overline{y}$. The explanation is proved to be valid as long as constraints compute valid explanations as they perform a valid pruning. Note that the structure of the dual is used through the explanation algorithms embedded within constraints. In fact, the computation of explanations is *lazy*[2]. Therefore, such an inference dual provides an arbitrary[3] dual solution but not necessarily the optimal one.

3.2 Decomposition scheme

One of the key point of Benders decomposition is to be able to derive a master problem that provides a valid lower bound for the original P. We used the following master problems for initial problems P_y, P_0 and P_{xy} :

$$\begin{array}{ll|ll|ll} MP_y: & \text{Min } f(y) & MP_0: & \text{Min } 0 & MP_{xy}: & \text{Min } r(y) = relax(f(x,y)) \\ & \text{s.t} : Ct^i(x,y) & & \text{s.t} : Ct^i(x,y) & & \text{s.t} : Ct^i(x,y) \\ & y \in D_y & & y \in D_y & & y \in D_y \end{array}$$

One can notice here that MP_y provides a valid lower bound as it is a relaxation of P_y. It is also the case for P_0 as it is a satisfaction problem. However, it is not true in the general case of P_{xy} where a specific master problem must be designed. In fact, a new objective function called $r(y)$ defined on y variables and providing a lower bound has to be defined by the user.

There are some cases where the original function is itself a relaxation (e.g. a coloring problem) but in a generic case, the master problem take the form of a feasibility problem where the cuts added can be seen as $expl(z \geq z^*) \Rightarrow z \geq z^*$.

4 A Benders decomposition algorithm for CP

Figure 1 presents our algorithm[4]. It has been implemented as a library of the Java version of the `PaLM` solver embedded within the `choco` (see `http://choco.sf.net`) constraints solver. The standard *CP* model is only enriched by indicating for each variable the problem to which it belongs (the master or the index of the sub-problem).

[2] Not all possible explanations are computed when removing a value for scalability reasons. Only the one corresponding to the solver actual reasoning is kept.

[3] It is also the case for linear duality as any dual solution is a bound for the primal.

[4] Line 8 is used in the case of $P_0^{'}$ and $P_y^{'}$ whereas lines 5, 11, 13, and 14 concerns P_{xy}. The case of $P_{xy}^{'}$ is not yet included.

```
input : an initial solution to the master problem $\overline{y}$,
(1)  begin
(2)    repeat
(3)      $Cut = \emptyset$
(4)      for each sub-problem $spb_k$ do
(5)        $P_{xy}$ : update upper bound of $spb_k$ with computeUb(k) using $\{\overline{z}, \overline{sz_i}, \forall i < k\})$
(6)        solve $spb_k$ on $(\overline{y}, x_k)$ to optimality
(7)        add its inconsistency (if $spb_k$ is infeasible)/optimality explanation to $Cut$
(8)        $P_0^{'}, P_y^{'}$ : $spb_{k+1} = \bigcup_{i \leq k, i > k'} spb_i$, with $k'$, the last infeasible sub-problem.
(9)      endfor
(10)     if $(Cut \neq \emptyset)$ then
(11)       $P_{xy}$ : $Cut$ = computeCut($Cut$)
(12)       add all explanations $\in Cut$ to the master problem
(13)       $P_{xy}$ : update the upper bound of $z$ with computeUb(0) using $\{\overline{z}, \overline{sz_1}, \ldots, \overline{sz_n}\})$
(14)       $P_{xy}$ : store $(\overline{x}, \overline{y})$ if it is an improving solution
(15)       solve the master problem to optimality
(16)     endif
(17)   until the master problem is infeasible $\bigvee Cut = \emptyset$
(18)   $P_y$, $P_0$, $P_0^{'}$, $P_y^{'}$ : the solution $(\overline{y}, \overline{x})$ is optimal if $Cut = \emptyset$ otherwise, P is infeasible.
(19)   $P_{xy}$ : the solution $(\overline{y}, \overline{x})$ is the optimal solution of $P$ otherwise P is infeasible.
(20) end
```

Fig. 1. A Generic Benders algorithm for P_0, P_y ($P_0^{'}$, $P_y^{'}$) and P_{xy}

4.1 Specific handling for problems P_0 and P_y

P_0 and P_y are closely related because they both use satisfaction problems as sub-problems. Backjumping algorithms are used to compute explanations (to provide dual informations) on the sub-problems. Moreover, the use of backjumping for the master is possible for P_0 (which is a traditional *CSP*) and allows the partial avoidance of thrashing on the master problem when adding the cuts. This is a response to Thorsteinsson [5] concerns about possible significant overhead due to redundant computations. Concerning approximated structures: at any iteration k, the next sub-problem $k+1$ considered is chosen according to the rule described line 8. So if one sub-problem is consistent, the next one starts from its solution and consider for branching the variables of both problems. Such a strategy hopes to benefit from the relative independency of sub-problems (it does not imply any overhead compared to solving one single sub-problem) to derive disjoint cuts. There is obviously a compromise between the time spent for solving sub-problems and the accuracy of the retrieved information.

4.2 Specific handling for problem P_{xy}

To keep isolated sub-problems, we do not add the objective function as a constraint which could propagate from one sub-problem to another. Instead, we provide a way to compute the bound of one problem according to other known bounds (master and slaves) with an empty explanation. So the propagation is done *at hand* to only incriminate the master problem solution using:

– *computeCut(Explanation[] expls)* (line 11): computes the explanation(s) to be added to the master according to the objective function. A sum would lead to a union among explanations for example;
– *computeUb(int k)* (line 5,13): computes an upper bound on z_k according to $\overline{y}$ and known z_i with $i < k$. In the case $k = 0$ (the master problem) it computes the upper bound of the overall objective function z if every SP_k was feasible.

At each iteration, a lower bound is obtained once the master problem has been solved. The algorithm stops once the lower bound meets the upper bound computed after the slaves. One can notice that the upper bound does not necessarily follow a decreasing trend whereas the lower bound is only growing ensuring the termination of the algorithm as long as variables have finite domains.

5 Conclusion

We have investigated in this paper how to derive logic Benders cuts using an explanation based framework for Constraint Programming. Accuracy of cuts using explanations is nevertheless questionnable. Indeed, remaining sub-problems are not polynomial (compared to a traditional MILP approach for Benders and assuming that LP is polynomial) and explanations constitute a weaker cut as a lazy computation is used. First experimental results using structured random binary problems show that Benders becomes advantageous in case of hard sub-problems compared to a branching using the same structure information within a backjumping algorithm. Moreover, we believe that the presence of subset of variables exhibiting a strong impact over the whole problem could be efficiently used by such an approach. Our next step is to apply the technique on hard academical problems and we are currently investigating how hard latin square instances could be decomposed.

References

1. J. F. Benders. Partitionning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
2. A. M. Geoffrion. Generalized Benders Decomposition. *Journal of Optimization Theory And Practice*, Vol. 10, No. 4, 1972.
3. J.N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96:33–60, 2003.
4. Vipul Jain and I. E. Grossmann. Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on Computing*, 13:258–276, 2001.
5. Erlendur S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *CP'01*, 2001.
6. John N. Hooker A Hybrid Method for Planning and Scheduling. In *CP'04*, pages 305–316, 2004.
7. T. Benoist, E. Gaudin, and B. Rottembourg. Constraint programming contribution to benders decomposition: A case study. In *CP'02*, pages 603–617, 2002.
8. H. Cambazard, P. E. Hladik, A. M. Déplanche, N. Jussien, and Y. Trinquet. Decomposition and learning for a real time task allocation problem. In *CP'04*, pages 153–167, 2004.