

Benders decomposition in 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

Abstract. Recent work have exhibited specific structure among combinatorial problem instances that could be used to speed up search or to help users understand the dynamic and static intimate structure of the problem being solved. Several Operations Research approaches apply decomposition or relaxation strategies upon such a structure identified within a given problem. This paper presents how Benders decomposition could be adapted to constraint programming when specific relationships between variables are exhibited. It discusses the way a decomposition framework could be embedded in constraint solvers, taking advantage of structures for a non expert user in a generic way. To achieve the interaction between structures, it explores the possibility of deriving logic Benders cuts using an explanation based framework for Constraint Programming.

1 Introduction

Benders decomposition [2] has been successfully used in numerous situations in operations research. It is often introduced as a basis for models and techniques using the complementary strengths of constraint programming and optimization techniques. Hybridation schemes have appeared recently and provided interesting computational results [13, 6, 12, 14, 3]. They have been extended [5, 15] to take into account other kinds of sub-problems and not only the classical programming linear ones. However, decomposition has never been used to our knowledge in a pure constraint programming approach.

Real industrial problems often exhibit different sets of variables playing distinct business roles: they are in fact often structured problems. This paper discusses the way a decomposition framework could be embedded in constraint solvers, taking advantage of structures for a non expert user in a generic way. We explore the possibility of deriving logic Benders cuts using an explanation based framework for Constraint Programming. We evaluate how the use of explanations could take advantage of a master-slaves relationship detected in the problem and describes Benders decomposition as a nogood recording strategy in Constraint Programming. Moreover we propose to apply this technique in the case of approximated structures for the sub-problems.

Section 2 will introduce explanation based constraint programming as well as the basis of Benders decomposition. Section 3 and 4 will then focus on the use of explanation as Benders cuts and the main algorithm implemented on the PaLM framework. First practical results will be presented in Section 5.

2 Context

2.1 Explanations for constraint programming

An explanation records information to justify a decision of the solver as a reduction of domain or a contradiction. It is made of a set of constraints C' (a subset of the original constraints of the problem) and a set of decisions dc_1, dc_2, \dots, dc_n taken during search (assignments are typical decision constraints, but other more generic constraints can be considered). An explanation of the removal of value a from variable v , $expl(x \neq a)$ will be written as follows:

$$C' \wedge dc_1 \wedge dc_2 \wedge \dots \wedge dc_n \Rightarrow v \neq a$$

When a domain is emptied, a contradiction is identified. An explanation for this contradiction is computed by uniting each explanation of each removal of value of the variable concerned. Intelligent backtracking algorithms that question a relevant decision appearing in the conflict are then conceivable [4]. By recording a relevant part of the explanations involved in conflicts, a learning mechanism can be implemented [9]. Notice that the set of decisions is also called a nogood. It represents a partial assignment that can not be extended to a solution. On the previous definition, $dc_1 \wedge dc_2 \wedge \dots \wedge dc_n \wedge v = a$ is the corresponding nogood.

2.2 Principles of Benders decomposition

Benders decomposition can be seen as a form of *learning from mistakes*. It is a solving strategy that uses a partition of the variables of the problem into two sets: x, y . The strategy can be applied to a problem of this general form:

$$\begin{aligned} P : & \text{Min } f(y) + cx \\ \text{s.t.} : & g(y) + Ax \geq a \text{ with } : y \in D, x \geq 0 \end{aligned}$$

A master problem considers only a subset of variables y (often integer variables, D is a discrete domain). A sub-problem (SP) tries to complete the assignment on x . If it is possible, the problem is solved, but if not, a cut (rejecting at least the current assignment on y) is produced and added to the master problem: it is called a Benders cut.

This cut has the form $z \geq h(y)$ (z represents the objective function $-z = f(y) + cx$) and constitutes the key point of the method, it is inferred by the dual of the sub-problem. Let us consider an assignment y^* given by the master, the sub-problem (SP) and its dual (DSP) can be written as follows:

$$\begin{array}{ll} \text{SP : Min } cx & \text{DSP : Max } u(a - g(y^*)) \\ \text{s.t } Ax \geq a - g(y^*) \text{ with } : x \geq 0 & \text{s.t } uA \leq c \text{ with } : u \geq 0 \end{array}$$

Duality theory ensures that $cx \geq u(a - g(y^*))$, $u(a - g(y^*))$ is therefore a lower bound of cx . As feasibility of the dual is independent of y^* , $cx \geq u(a - g(y))$ and the inequality $f(y) + cx \geq f(y) + u(a - g(y))$ is valid, leading to the Benders cut: $z \geq f(y) + u(a - g(y))$. Moreover, according to duality, the optimal value of u^* maximizing $u(a - g(y^*))$ corresponds to the same optimal value of cx . Even if the cut is derived from a particular y^* , it is valid for all y and excludes a large class of assignments which share common characteristics. The number of solutions to explore is reduced and the master problem can be written at the k^{th} iteration:

$$\begin{aligned} \text{MP : Min } z \\ \text{s.t : } z \geq f(y) + u_i^*(a - g(y)) \quad \forall i < k \end{aligned}$$

From all of this, it can be noticed that duals variables or multipliers¹ need to be defined to apply the decomposition. However, a generalized scheme has been proposed in 1972 by Goeffrion [15]. Hooker and Ottosson [5] proposed also to overcome this limit and to enlarge the classical notion of *dual* by introducing an *inference dual* available for all kinds of sub-problems. They refer to a more general scheme and suggests a different way of thinking about duality: a Benders decomposition based on *logic*. Duality means now to be able to produce a proof, the logical proof of optimality of the sub-problem and the correctness of cuts inferred. In classical Benders decomposition, this proof is established thanks to classical duality theorems. 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:

$$\begin{aligned} P : \text{Min } obj \\ \text{s.t : } Ct(x, y) \\ \text{with : } x \in D_x, y \in D_y \end{aligned}$$

$Ct(x, y)$ denotes a set of constraints on variables x, y and obj can be equal to $\{f(x, y), f(y), 0\}$ (respectively, an objective function on the whole set of variables, an objective function restricted to the y variables and a satisfaction problem). 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, there should be as small and independant 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 or approximate structure. P is supposed to have an ideal

¹ Referring to linear programming duality.

structure whereas it is denoted P' in case of such an approximated structure over the x variables (so we get in the same way $\{P'_{xy}, P'_y, P'_0\}$). Master problem and sub-problems have then the generic form :

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

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 of the master, the added Benders cut ensures that :

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 \bar{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 . While solving the sub-problem, classical cuts on the objective function are added $sz < sz^*$. Once the optimal solution has been found, the problem becomes inconsistent with the previous cut and the solver provides an explanation where the set of decision constraint taken at the sub-problem level is empty. This explanation give a set of constraints (original constraints $Ct(x, y)$ and decision constraints $dc(y)$) to explain why the sub-problem can not be extended to a better solution than sz^* written as : $expl(sz \geq sz^*)$.

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 problem P whereas the complete set excludes only the current assignment. The explanation is proved to be valid as long as each constraint computes a valid explanation in the same way that it performs 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, the first explanation is taken even if several explanations exist. One cannot look for the minimal explanation for evident scalability reasons. Therefore, such an inference dual provides an arbitrary³ dual solution but not necessarily the optimal one. Obviously, the success of such an approach depends on the degree to which accurate explanations can be computed for the constraints of the sub-problem.

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

³ This can also be accounted for linear duality where any dual solution is a bound for the primal problem.

3.2 Decomposition scheme

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

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

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. Consider for example a graph coloring problem where the number of different colors must be minimized. In this case, one can directly use the original objective as a relaxation.

$$\begin{array}{l} P1 : \text{Min } NbDifferentValues(x, y) \\ \text{s.t : } x_i \neq x_j \forall (i, j) \in G \end{array}$$

We have obviously $NbDifferentValues(y) \leq NbDifferentValues(x, y)$ and we can use $NbDifferentValues(y)$ as a valid objective for the master problem. In a generic case, the master problem take the form of a feasibility problem where the cuts added can be seen as $expl(z \geq z^*) \Rightarrow z \geq z^*$

We also consider the following example which will be developed later with an optimisation function given as a linear combination:

$$\begin{array}{l} P2 : \text{Min } y_1 + y_2 + 4x_1 + 2x_2 + 5x_3 + x_4 \\ \text{s.t : } |y_1 - y_2| < 2 \\ 5x_1 + x_2 + 2y_1 + y_2 \geq 5 \\ 3x_3 + 2x_4 + y_2 \geq 4 \\ x_1 \neq y_2 \\ occurrence(0, \{x_3, x_4, y_2\}) = 1 \\ occurrence(0, \{x_3, x_4, y_1\}) = 1 \\ y_1, y_2, x_1, x_2, x_3, x_4 \in [0, 5] \end{array}$$

In this case again, the use of $\text{Min } y_1 + y_2$ gives a correct master problem.

4 A Benders decomposition algorithm for constraint programming

Figure 1 presents our algorithm. It has been implemented as a library of the Java version of the PaLM [8] solver embedded within the `choco`⁴ constraints solver. The

⁴ See <http://choco.sf.net>.

standard CP model taken as input by the solver is only enriched by indicating for each variable the problem to which it belongs (the master or the index of the sub-problem). Classical branching can be specified for the variables of each sub-problem. The cuts are managed as a single global constraint that filters the last non instantiated variable of each cut. Improvements can be expected from that side by using a GAC algorithm for example. Note that line 10 of the algorithm is used in the case of approximated structures for P_0 and P_y whereas lines 6, 8, 16, and 17 concerns P_{xy} .

Benders algorithm :

```

(1) input : an initial solution to the master problem  $\bar{y}$ ,
(2) begin
(3)   repeat
(4)      $Cut = \emptyset$ 
(5)     for each sub-problem  $spb_k$  do
(6)        $P_{xy}$  : update upper bound of  $spb_k$  with computeUb(k) using  $\{\bar{z}, \bar{s}z_i, \forall i < k\}$ 
(7)       solve  $spb_k$  on  $(\bar{y}, x_k)$  to optimality
(8)        $P_{xy}$  : add its optimality explanation  $expl_k$  to  $Cut$ 
(9)        $P_0, P_y$  : if  $spb_k$  is infeasible, add its inconsistency explanation  $expl_k$  to  $Cut$ 
(10)       $P'_0, P'_y$  :  $spb_{k+1} = \bigcup_{i \leq k, i > k'} spb_i$ , with  $k'$ , the last infeasible sub-problem.
(11)     endif
(12)   endfor
(13)   if ( $Cut \neq \emptyset$ ) then
(14)      $Cut = \text{computeCut}(Cut)$ 
(15)     add all explanations  $\in Cut$  to the master problem
(16)      $P_{xy}$  : update the upper bound of  $z$  with computeUb(0) using  $\{\bar{z}, \bar{s}z_1, \dots, \bar{s}z_n\}$ 
(17)      $P_{xy}$  : store  $(\bar{y}, \bar{x})$  if it is an improving solution
(18)     solve the master problem to optimality
(19)   endif
(20) until the master problem is infeasible  $\vee Cut = \emptyset$ 
(21)  $P_y, P_0, P'_y, P'_0$  : the solution  $(\bar{y}, \bar{x})$  is optimal if  $Cut = \emptyset$  otherwise, P is infeasible.
(22)  $P_{xy}$  : the solution  $(\bar{y}, \bar{x})$  is the optimal solution of  $P$  otherwise P is infeasible.
(23) end

```

Fig. 1. A Generic Benders algorithm for 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 for the sub-problems to compute explanations (we do not use them to perform dynamic backtracking, we therefore do not maintain the explanation network among constraints⁵) to provide dual informations on the sub-problems. Moreover, the use of backjumping for the master is possible for P_0 (which is a traditionnal CSP) and allows the partial

⁵ This is possible with the new version of `choco` which contains a `mac-cbj` mode.

avoidance of thrashing on the master problem when adding the cuts. This is a response to Thorsteinsson [13] concerns about possible significant overhead due to redundant computations. Termination of the algorithm can be seen at line 22: an infeasible master problem for P_0 and P_y means that the whole problem is infeasible, otherwise an optimal or feasible solution has been found.

One word has to be said about approximated structures: at any iteration, once the master problem and k sub-problems have been solved, the next sub-problem $k + 1$ considered is done according to the following rule. Let $k' < k$ be the index of the last infeasible sub-problem $spb_{k'}$, spb_{k+1} is then defined as : $spb_{k+1} = \bigcup_{i \leq k, i > k'} spb_i$. 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 does not imply any overhead compared to solving one single sub-problem (it only changes the variable ordering) but hopes to benefit from the relative independency of sub-problems to derive disjoint cuts. There is obviously a compromise between the time spent for solving sub-problems and the quality of the retrieved information. For example, as soon as one cut has been derived, it seems reasonable to limit the search for other cuts to the sub-problems themselves.

4.2 Specific handling for problem P_{xy}

As we want 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 for each sub-problem. The interest of this implementation is to allow to solve separately each sub-problem in order to get at the end several valid explanations for the master. Take the example of P_1 , one could stop once we get a first sub-problem with a coloring number higher than our current upperbound, or one can continue solving the next sub-problem to derive disjoint or maybe more accurate explanations. The key is to take advantage of the structures and to find from all subproblems a cut that will avoid as more thrashing as possible on the master.

- *computeCut(Explanation[] expls)* (line 14): computes the explanation(s) to be added to the master according to the objective function. Consider for example the problem P_2 , it can be implemented as a union over explanations of sub-problems whoses sum of their optimum exceed the best known upperbound; For P_1 , as a subset of non including explanations of sub-problems exceeding the coloring number of the master. Note that it can be overridden by a user also for generic problems P_0 and P_y to implement a specific heuristic to manage cuts such as keeping only the smallest ones or enough disjoint ones.
- *computeUb(int k)* (line 16): computes an upper bound on z_k of sub-problem k according to \bar{y} and the currently 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 all bounds of sub-problems are known (every sub-problem 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. Note that when considering the generic problems P_{xy} , the sub-problems have to be solved to optimality to provide the explanations.

Let us use our example problem P_2 to illustrate how algorithm 1 works. Notice that once y_1 and y_2 are instantiated, we get the following sub-problems :

$$\left. \begin{array}{l} \text{SP1 : Min } sz_1 = 5x_3 + x_4 \\ 3x_3 + 2x_4 \geq 4 - \bar{y}_2 \\ \text{occurrence}(0, \{x_3, x_4, \bar{y}_2\}) = 1 \\ \text{occurrence}(0, \{x_3, x_4, \bar{y}_1\}) = 1 \end{array} \right| \begin{array}{l} \text{SP2 : Min } sz_2 = 4x_1 + 2x_2 \\ 5x_1 + x_2 \geq 5 - 2\bar{y}_1 - \bar{y}_2 \\ x_1 \neq \bar{y}_2 \end{array}$$

Iteration 1	$(\bar{y}_1, \bar{y}_2) = (0, 0)$ $sz_1^* = 6$ $sz_2^* = 4$ $z = 10$	$\text{expl}(sz_1 \geq 6) = \{y_2 = 0\}$ $\text{expl}(sz_2 \geq 4) = \{y_2 = 0\}$ $y_1 + y_2 < 10, y_2 \neq 0$ added
Iteration 2	$(\bar{y}_1, \bar{y}_2) = (1, 1)$ $sz_1^* = 2$ $sz_2^* = 4$ $z = 8$	$\text{expl}(sz_1 \geq 2) = \{y_2 = 1\}$ $\text{expl}(sz_2 \geq 4) = \{y_1 = 1, y_2 = 1\}$ $y_1 + y_2 < 8, y_1 \neq 1 \vee y_2 \neq 1$ added
Iteration 3	$(\bar{y}_1, \bar{y}_2) = (2, 1)$ $sz_1^* = 2$ $sz_2^* = 0$ $z = 5$	$\text{expl}(sz_1 \geq 2) = \{y_2 = 1\}$ $\text{expl}(sz_2 \geq 0) = \{y_1 = 2, y_2 = 1\}$ $y_1 + y_2 < 5, y_1 \neq 2 \vee y_2 \neq 1$ added
Iteration 4	$(\bar{y}_1, \bar{y}_2) = (1, 2)$ $sz_1^* = 1$ SP2 infeas $z = 5$	$\text{expl}(sz_1 \geq 2) = \{y_2 = 2\}$ $\text{expl}(sz_2 \geq 0) = \{y_1 = 1, y_2 = 2\}$ $y_1 \neq 1 \vee y_2 \neq 2$ added
Iteration 5	$(\bar{y}_1, \bar{y}_2) = (2, 2)$ $sz_1^* = 1$	$\text{expl}(sz_2 \geq 2) = \{y_2 = 2\}$ $y_2 \neq 2$ added

At iteration 1, one can notice that two explanations exist for $sz_1 \geq 6$: $\{y_2 = 0\}$ or $\{y_1 = 0\}$. Each one is sufficient to explain that $sz_1 \geq 6$. The explanation retrieved will depend of the propagation order of the two occurrence constraints. Let us say that $\text{occurrence}(0, \{x_3, x_4, y_2\})$ has been propagated first, removing the lower bound 0 of x_3 and x_4 . After solving the sub-problem, we know that z cannot be less than 10 as long as y_2 is null. For iteration 5, the upper bound computed with $\text{computeUb}(2)$ is negative and the explanation for $SP2$ is empty. After iteration 5, the master problem becomes inconsistent and the algorithm terminates with the optimal solution $(y_1, y_2, x_3, x_4, x_1, x_2) = (2, 1, 0, 2, 0, 0)$.

5 The interest of decomposition in Constraint Programming

We describe in this section several applications and preliminary experimental results on the decomposition.

5.1 Accuracy of cuts on random structured binary problems

We study in this section the interest of taking advantage of a structure in the generic context of classical random binary problems. Random instances are usually characterized by the tuple $\langle n, d, p_1, p_2 \rangle$ (we use the classical B model [1]) where n is the number of variables, d the unique domain size, p_1 the density of the constraint network and p_2 the tightness of the constraints. To build structured instances, we use the following model $\langle nm, ns, nbs, p_1, p_m, p_s, p_{ms} \rangle$, where nm , ns are the number of variables of master and slaves, nbs , the number of slaves and p_m , p_s , p_{ms} the tightness of master, slaves and master-slave relationship. Figure 2 shows the results on structured problems RP_1 of the form $\langle 10, 10, nbs, 50\%, p_2, p_2, p_2 \rangle$. So the same tightness is used for both master, slaves and master-slaves relationship. Each point represents the average computation time and number of nodes explored for our Benders decomposition algorithm (**Benders** from now on) and a classical search performed on the whole problem using the **mindom** heuristic (**mindom** from now on) over the same series of 100 instances randomly generated at the phase transition of RP_1 .

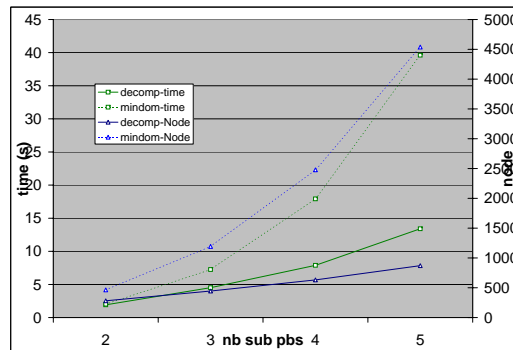


Fig. 2. Time and number of nodes needed for **mindom** and **Benders** (decomp) on structured problem $\langle 15, 6, nbs, 50\%, p_2, p_2, p_2 \rangle$ at the transition phase for p_2 with nbs varying from 2 to 5 sub-problems.

As expected, the gap between the two techniques increases quickly with the degree to which the whole problem is structured. Surprisingly, a relative small number of sub-problems (compared to similar random experiments in pure linear programming of [5]) is needed to make **Benders** really more efficient. However this can only be due to the good ordering performed by the decomposition. So,

we then tried to isolate the effect of the decomposition itself, *i.e* the learning of cuts and the study of several parts of the problem before coming back to the master. We apply a branching strategy denoted as *ad hoc branching* that uses the mindom heuristic among the master variables and each sub-problem in the same order as **Benders**. It is sufficient on the previous benchmark to close the gap between the two strategies.

So we investigated what kind of relationships could make our Benders decomposition more efficient. We discovered that the *ad hoc* strategy seems to experience difficulties just before the phase transition of the sub-problems (a tightness around 82-83% for 6 variables). Hard sub-problems seems to make **Benders** more efficient. We adapt the previous benchmark and fix this time the sub-problem tightness to 82% (the phase transition of sub-problems) to get problems of the following form $\langle 15, 6, nbs, 50\%, p_2, 82\%, 3\% \rangle$. We then report in Figure 3 the number of unsolved problems for both *ad hoc* and **Benders** strategy for different tightness of the master and number of sub-problems within a time limit of 2 minutes. The master problem becomes in fact sometimes itself difficult only due to the interaction with slaves without instantiating them.

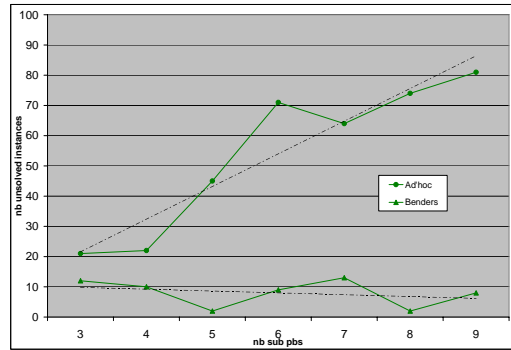


Fig. 3. Number of unsolved instances for **Benders** and *ad hoc strategy* on structured problems with sub-problems at the phase transition.

Benders strategy allow to solve more and more instances as the number of sub-problems grow. *Ad hoc* branching is in fact more subject to thrashing on the master tree search as it tries to solve each sub-problem one after another without looking at the rest of the problem which can sometimes help pruning efficiently the master as Benders do.

5.2 Application to real time scheduling

The approach described in this paper has been firstly applied in [3] for a real time scheduling problem. It proposed at that time to implement a dedicated *logical* duality to infer nogoods, tried to enforce the constraint model and finally performed an incremental resolution of the master problem using a dynamic

backtracking algorithm for the master. The problem was to assign periodic tasks to processors in the context of fixed priorities preemptive scheduling (a task is periodically activated and can be preempted by a higher priority task). Real-time schedulability enforces the duration between the activation date of a task and its completion time to be bounded by its deadline. It guarantees that relative deadlines of tasks will be met.

This problem fits in the generic problems P_0 framework. Moreover, the set of variables x for the sub-problem is empty. Instead of providing a filtering algorithm within a global constraint for real-time schedulability, we attempted to derive accurate explanations for the unschedulability of each processor and therefore built a specific instance of the decomposition framework described here. So the partition of the problem was rather made among the constraints of the problem and the sub-problems considered had to evaluate the real-time schedulability of a set of tasks (jobs) on a processor (machine). In a sense, it was not far from the hybrid framework *Branch and Check* of [13] which consists in checking the feasibility of a delayed part of the problem in a sub-problem. In case of failure, it returned a minimal (regarding inclusion) subset of tasks responsible for this unschedulability. The computation of this minimal set was obtained by coupling analytical techniques from real time scheduling with a conflict based algorithm, QuickXplain [10]. It is a good specific example of application as sub-problems were polynomials and the cuts generated were quite efficient because we could compute the minimal ones. The whole scheme could take advantage of polynomiality of sub-problems as well as symmetry among processors (a cut derived on one processor was also valid for any processor of the system).

6 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 was nevertheless questionable. 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. We therefore tried to show on random problems that explanations could provide a relevant cut in case of hard sub-problems. We attempted to demonstrate how such a decomposition framework could be embedded in constraint solvers to take advantage of explicit or implicit structures within a given problem/instance. Moreover, we believe that the presence of *backdoors* [11] or subset of variables exhibiting a strong impact over the whole problem could be explicitly and 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.

Numerous perspectives exist. One promising application not described in Section 5 is Stochastic Constraint Programming [17] as Benders decomposition is used as an efficient way of tackling scenario-based approach for those problems. A current limitation for this technique is to be able to consider any kind of recourse

and not only linear recourse. Concerning the whole scheme, the management of a large base of explanations could be improved considering that lot of variables are shared among explanations. Finally, indicators based on impact computation [16] could be used to automatically discover relevant master and slaves problems.

References

1. D. Achlioptas, L. Kirousis, E. Kranakis, D. Krizanc, M. Molloy, and Y. Stamatiou. Random constraint satisfaction: a more accurate picture. In *Proceedings CP 1997*, pages 121–135, Linz, Austria, 1997.
2. J. F. Benders. Partitionning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
3. Hadrien Cambazard, Pierre-Emmanuel Hladik, Anne-Marie Déplanche, Narendra Jussien, and Yvon Trinquet. Decomposition and learning for a real time task allocation problem. In *Proceedings CP 2004*, pages 153–167, 2004.
4. Bruno de Backer and Henri Béringier. Intelligent Backtracking for *CLP* Languages: An Application to *CLP(R)*. in Proceedings of ILPS'91: Proceedings International Logic Programming Symposium, pages 405–419, 1991.
5. J.N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96:33–60, 2003.
6. 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.
7. Narendra Jussien. *The versatility of using explanations within constraint programming*. Habilitation thesis, Université de Nantes, France, 2003. also available as RR-03-04 research report at École des Mines de Nantes.
8. Narendra Jussien and Vincent Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pages 118–133, Singapore, September 2000.
9. Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, July 2002.
10. Ulrich Junker. Quickxplain: Conflict detection for arbitrary constraint propagation algorithms. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints (CONS-1)*, Seattle, WA, USA, August 2001.
11. Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proceedings IJCAI 2003*, 2003.
12. J.N. Hooker, G. Ottosson, E. S. Thorsteinsson, and H. Kim. A scheme for unifying optimization and constraint satisfaction methods. *Knowledge Engineering Review, special issue on AI/OR*, 15(1):11–30, 2000.
13. Erlendur S. Thorsteinsson. Branch-and-check: A hybrid framework integrating mixed integer programming and constraint logic programming. In *CP'01*, 2001.
14. T. Benoist, E. Gaudin, and B. Rottembourg. Constraint programming contribution to benders decomposition: A case study. In *CP'02*, pages 603–617, 2002.
15. A. M. Geoffrion. Generalized Benders Decomposition. *Journal of Optimization Theory And Practice*, Vol. 10, No. 4, 1972.
16. H. Cambazard, N. Jussien. Identifying and exploiting problem structures using explanation-based constraint programming. International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'05), 2005.

17. S. Manandhar, S. A. Tarim, T. Walsh. Scenario-Based Stochastic COntstraint Programming. Proceedings of IJCAI-2003, Acapulco, Mexico, 2003, 257-262.