Identifying and exploiting problem structures using explanation-based constraint programming

Hadrien Cambazard and Narendra Jussien
École des Mines de Nantes – LINA CNRS FRE 2729
4 rue Alfred Kastler – BP 20722 – F-44307 Nantes Cedex 3 – France
`email:` *{Hadrien.Cambazard,Narendra.Jussien}@emn.fr*

Abstract. Identifying structures in a given combinatorial problem is often a key step for designing efficient search heuristics or for understanding the inherent complexity of the problem. Several Operations Research approaches apply decomposition or relaxation strategies upon such a structure identified within a given problem. The next step is to design algorithms that adaptively integrate that kind of information during search. We claim in this paper, inspired by previous work on impact-based search strategies for constraint programming, that using an explanation-based constraint solver may lead to collect invaluable information on the intimate dynamically revealed and static structures of a problem instance. Moreover, we discuss how dedicated OR solving strategies (such as Benders decomposition) could be adapted to constraint programming when specific relationships between variables are exhibited.

1. Introduction

Generic search strategies for solving combinatorial optimisation problems represent the Holy Grail for both Operations Research (OR) and Constraint Programming (CP) people. Several tracks are now explored: dynamically adapting the search strategy, identifying specific structures in a given instance in order to speed up search, etc. Whatever the technique, the key point is to be able to identify, understand and use the intimate structure of a given combinatorial problem instance (Gomes et al, 1997; William et al, 2003a; William et al, 2003b). For example, Bessière *et al.* (2001) proposed to take into account variable neighborhood; more recently, Refalo (2004) defined impact-based solving strategies for constraint programming. These new techniques, taking into account propagation, dynamically use the structure of the solved problem.

In this paper, we attempt to identify, differentiate and exploit problem structure that is revealed during the course of the solution algorithm. We focus on structure that appears in the form of subsets of variables that play a specific role in the problem. To this end we define several fine-grained measures of the impact of fixing the values of the variables in these subsets. The goal is to:

- identify structure that is not initially visible;

- design new generic techniques for guiding the search;
- pave the way for the use of impact analysis in such decomposition-based methods as Benders decomposition.

Our new impact measures are made possible by the use of an explanation-based constraint solver that provides inside information about the solver embedded knowledge gathered from the problem. This information represent some kind of trace of the inferences made during propagation and therefore implicitly outlines relationships between variables.

The paper is organized as follows: Section 2 introduces the basis and motivations of our work. Several impact measures and associated graphs are presented in Section 3 distinguishing their respective ability to reflect dynamically revealed and static structures on a concrete example. Finally, as we believe that the detection of hidden structures can be explicitly used into CP, we start to show the interest of those such structures as a guide for searching as well as the design of a dedicated resolution strategy inspired from a logic-based decomposition.

2. Search strategies for structured problems

Efficient constraint programming search strategies exploit specific aspects or characteristics of a given (instance of a) problem. In OR, relaxation or decomposition strategies exploit the fact that part of the problem can be considered as a *classical* problem (such as compatible or optimal flow problems, shortest path problems, knapsack problems, etc.). This aspect of the problem is often called *structure*.

A problem is more generally said to be *structured* if its components (variables[1] and/or constraints) do not all play the same role, or do not have the same importance within the problem. In such a problem, the origin of the complexity relies on the different behavior (or impact) for specific components of the problem. One of the main difficulties in identifying structures in problems is that it is not always evident until one begins to solve the problem. The interplay between a given instance and the search algorithm itself may define or help to exhibit a hidden *structure* within the problem. We call it a *dynamically revealed structure*. It is related to initial choices that direct search to wrong directions as well as new relationships due to the addition of constraints during search.

[1] In the following, we will focus our study only on variables as components inducing a structure.

Structures are characterized using various notions: for example, *backbones* (Monasson et al, 1999) are variable assignments that appear in every solutions. *Backdoors* recently introduced in (William et al, 2003b) represent an interesting concept to characterize hidden structures within a problem. Informally, it can be defined as a subset of variables that encapsulate the whole combinatorics of the problem: once this subset instantiated, the remaining sub-problem can be solved very quickly. Several search strategies are based upon *backdoors*. The following two are central in this paper:

– Branching heuristics in CP attempt to early guide the search towards the *backdoor* variables as they try to perform choices that simplify the whole problem as much as possible. They are based on a simple idea: select a variable that leads to the possibly smallest search space and that raises contradictions as early as possible. This principle (often referred to as the *first fail* principle (Haralick and Elliot, 1980)) is often implemented by taking the current domain and degree of constraindness of the variables into account (see (Boussemart, 2004) or (Bessière et al, 2001) for variants). More recently, (Refalo, 2004) proposed to characterize the impact of a choice and a variable by looking at the average search space reduction caused by this choice (another way of identifying a *backdoor*);

– Benders decomposition (Benders, 1962) falls exactly within the range of backdoors techniques. It is a solving strategy based on a partition of the problem among its variables into two sets x, y. A master problem provides an assignment x^*, and a sub-problem tries to complete this assignment over the y variables. If this proves impossible, the sub-problem produces a *cut*[2] (a constraint) added to the master problem in order to prune this part of the search space on the x side. The interesting cuts are those which are able to prune not only the current x^* solution from the search space (this is mandatory) but also the largest possible class of assignments that share common characteristics with x^* which make them suboptimal or inconsistent for the same reason. This technique is intended for problems with a special structure. The master problem is based on a relevant subset of variables that generally verifies the two following assumptions:

    1. The resulting subproblem is easy. In practice, several small independent subproblems are used, making it easy to perform

[2] This cut is often referred to as the *Benders* cut.

the required exhaustive search in order to produce the Benders cut;

2. The Benders cut is accurate enough to ensure a quick convergence of the overall technique.

In such a decomposition, the master problem can be considered as a *backdoor* because, thanks to condition (1), once the master problem completely instantiated the subproblem can be solved efficiently. Moreover, if the remaining subproblem can be actually solved polynomially (this is referred to as *strong* backdoors), a powerful cut based on the minimal conflict can often be computed.

For decomposition techniques, the structure needs to be identified before search starts. Classical structure identification is made through an analysis of the constraint network. For example, it is common for solving graph coloring problems to look for maximal cliques in order to compute bounds or to add global constraints such as *all-different* (Régin, 1994) in order to tighten propagation on the problem. But, such an analysis only provides information on visible static structures. Nevertheless, hidden structures and dynamically revealed ones seem to be of very high interest for a lot of search strategies. Of course, their identification is at least as costly as solving the original problem. However, we believe that the propagation performed by the solver during search provides information that should lead to approximate those hidden structures. One way of exploiting that information is to use explanations (a limited, readable, and useable trace of the propagation process).

3. Identifying problem structure using explanations

Refalo (2004) introduced an impact measure with the aim of detecting choices with the strongest search space reduction. He proposes to characterize the impact of a decision by computing the Cartesian product of the domains (an evaluation of the size of the search space) before and after the considered decision. We want to go a step further by analyzing where this reduction occurs and how past choices are involved. We extend those measures into an impact graph of variables, taking into account both the effects of old decisions and their effective involvement in each inference made during resolution.

Our objective is to identify variables that maximally constrain the problem, or subsets of variables that have strong relationships and a strong impact upon the whole problem (just like a *backdoor*). We have focused our study on the following points:

- the impact or influence of a variable on the direct search space reduction;

- the impact of a variable inside a chain of deductions made by the solver even a long time after the variable has been instantiated;

- the region of the problem under the influence of a variable and the precise links between variables.

Explanations for constraint programming seem a relevant tool for providing such an information.

3.1. Explanations and Constraint Programming

Constraint programming is a generic search paradigm which relies on a tree-based exploration of the search space along with inference mechanisms (filtering algorithms) aimed at pruning as much as possible the search space. Search can be considered as a dynamic constraint addition technique. Such dynamic constraints will be referred to in the following as *decision constraints*. In this paper, we will consider assignment-like decision constraints: $x_i = a$ (the decision here amounts to choose a value for a variable). Each decision is propagated to the whole constraint network until a fix-point is achieved, a solution is found or a contradiction is identified. In this latter case, the search algorithms backtracks to the last choice point and choose another alternative. Explanations have been initially introduced to improve backtracking-based algorithms. They have been recently used for other purposes including dynamic constraint satisfaction problems and user interaction (Jussien, 2003).

DEFINITION 1. *An explanation records some sufficient information to justify an inference made by the solver (domain reduction, contradiction, etc.). It is made of a set of constraints C' (a subset of the set C of the original constraints of the problem) and a set of decisions dc_1, ..., dc_n taken during search. An explanation of the removal of value a from variable v will be written:*

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

Explanations computed by the constraint solver represent the logical chain of inferences made by the solver during propagation. In a way, they provide some kind of a trace of the behavior of the solver as any operation needs to be *explained*. In the following, we will refer to E as the set of explanations computed so far and E_i^{val} the set of explanations computed for all[3] removals of value *val* from the domain of variable i.

Explanations are computed on-the-fly by each constraint and stored at the variable level. They induce some additional time complexity (filtering algorithms need to integrate an explaining algorithm) and some additional space complexity ($O(nd)$ where n is the number of variables and d the maximum size of the domaine as at most one explanation is stored for any value[4]). Let $|e|$ be the size of an explanation e *i.e.* the number of constraints in e. Explanation e_1 will be considered as more precise than explanation e_2 if $|e_1| < |e_2|$. Indeed, the smaller an explanation, the more precise it is, as the number of necessary hypothesis to infer the associate value removal is reduced. Finally, the age a_e^d of a decision when computing explanation e is defined as the number of decisions taken since d when e is produced.

3.2. Characterizing impact

Refalo (2004) characterizes the impact of the decision $x_i = a$ as the search space reduction induced by this decision (following the *first fail* principle). Nevertheless, this reduction does not only occur when the decision is posted to the problem but also when other (future) deductions that are partially based on the hypothesis $x_i = a$ are made.

The use of explanations can provide more information on the real involvement of the decision in the reduction. A past decision $x_i = a$ has an effective impact (in the solver's point of view) over a value *val* of variable x_j if it appears in the explanation justifying its removal.

Our first measure is expressed as the number of times a decision occurs in a removal explanation for value *val* from variable x_j. The size of the explanation is also taken into account as it reflects directly the number of hypothesis required to deduce the removal. Limited hypothesis means higher possibility of occurrence for the associated removal. Hence, the relationship between associated variables should be stronger. I_0 is meant to measure this influence:

$$I_0(x_i = a, x_j, val) = \sum_{\{e \in E_j^{val}, x_i = a \in e\}} 1/|e| \tag{1}$$

From this basic measure we define several different others: first, two measures (I_1 and I_2) based on the solver's activity and explanations are introduced; second, a third one (I_3) taking into account the search

[3] A value can be removed from the domain of a variable several times during search. This is due to backtracking.

[4] This is an upper bound used to limit space occupation.

space reduction is proposed in order to identify static structures and to guide the search process.

As search and propagation are intricately related, it seems natural to normalize those measures *wrt* search.

- First, the impact is normalized *wrt* the number of times $|x_i = a|$ decision $x_i = a$ is taken. The idea is just not to overestimate frequently taken decisions:

$$I_1(x_i = a, x_j, val) = \frac{I_0(x_i = a, x_j, val)}{|x_i = a|}$$

- Another possible normalization is to consider the age a_e^d of a decision d when computing explanation e in order to decrease the impact of old decisions. We get:

$$I_2(x_i = a, x_j, val) = \sum_{\{e \in E_j^{val}, x_i = a \in e\}} \frac{1}{|e| \times a_e^{x_i = a}}$$

- As opposed to the approach used in (Refalo, 2004) (which computes an instantaneous impact) impact computation is scattered through the solving process each time an explanation is computed. I_3 therefore tries to identify recurrent search space reductions related to a given decision:

$$I_3(x_i = a, x_j, val) = \frac{I_0(x_i = a, x_j, val)}{|\{x_i = a \text{ active } \wedge val \in Dom(x_j)\}|}$$

$I_3(x_i = a, x_j, val)$ can be considered as the probability that the value val of x_j will be pruned if the decision $x_i = a$ is taken. This measure is therefore updated each time a new removal occurs and as long as $x_i = a$ is active. It takes into account the frequency as well as the proportion of the involvement of a decision within explanations of removals. Finally, I_1 tends to favor old decisions as opposed to I_2 who favors recent decisions. Section 3.4 will show the interest of these parameters for a final user and its comprehension of the solver's behavior.

Those impact measures highly depend on the effective exploration of the search space and techniques similar to those used in (Refalo, 2004) will be used to initialize them (propagation of each value in each variable's domain) and to sharpen them (using a *restart* protocol which restarts search in order to take into account from the root node, impacts computed in previous searches).

3.3. Variable relationships

We aim now at providing the final user a synthetic representation of the structure of the problem. This representation will be used to analyse and understand the problem and how the solver addressed it. Therefore, we would like to point out relations between variables. Thus, we need to aggregate the measures introduced above on all the values in the domain of a variable:

$$\forall \alpha \in [0,1,2], \qquad I_\alpha(x_i = a, x_j) = \sum_{val \in D(x_j)} I_\alpha(x_i = a, x_j, val)$$

I_3 needs to be handled separately as we need to relate the search space reduction of one variable on another one. We therefore consider the initial size of the domain:

$$I_3(x_i = a, x_j) = \frac{|D(x_j)| - \sum_{val \in D(x_j)} (1 - I_3(x_i = a, x_j, val))}{|D(x_j)|}$$

In this context, $1 - I_3(x_i = a, x_j, val)$ corresponds to the probability of presence of the value val of the variable x_j after taking $x_i = a$. We can now define the influence of a variable on another one in the following way:

$$I_\alpha(x_i, x_j) = \sum_{v \in D(x_i)} I_\alpha(x_i = v, x_j)$$

Relationships between variables can now be represented using an impact graph associated to each measure α. This graph is a valued oriented graph $GI_\alpha(X, E, W)$ where X is the set of variables in the problem and the weight for an arc $(x_i, x_j) \in E$ ($E = X \times X$) is defined as $I_\alpha(x_i, x_j)$. Each variables is a node of this graph and the weight of arc (x_i, x_j) represents the influence of x_i on x_j. The greater the weight, the greater the influence.

3.4. How to use impacts to analyze structures

We want now to show how the impact graph can be used to identify structures. We use here a concrete example and will show how information from the impact graph is analyzed to get information about the problem and its resolution.

3.4.1. *A particular instance*

We consider a random binary problem in which a structure is added by increasing the tightness of a subset of constraints in order to create several subsets of variables with strong relationships. Random instances are characterized by the tuple $< N, D, p_1, p_2 >$ (we use the classical B model (Achliptas et al, 1997)) 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 (the proportion of forbidden tuples) of the constraints. Here we consider $N = 30$, $D = 10$, $p_1 = 50\%$. We design three subsets of 10 variables whose tightness is $p_2 = 53\%$ while it is set to 3% in the remainder of the network.

The specific instance we chose here to illustrate our different measures is interesting because it seems harder to solve than expected for the `mindom` (Haralick and Elliot, 1980) classical variable selection heuristic (where the variable with the smallest current domain is chosen for instantiation). Using the different impact graphs introduced above, we would like to illustrate how several questions may be addressed when facing a problem instance:

- is it possible without any network analysis to identify the structure embedded within the instance ?
- why `mindom` is not performing as expected on this instance ? Is this due to the instance or to the heuristic itself ?

3.4.2. *Visualizing the impact graph*

Figures 1 to 4 show the impact graph GI of the 30 variables involved in our instance. We use here a matrix-based representation (Ghoniem et al, 2004): variables are represented both on the rows and columns of the matrix. The cell at the intersection of row i and column j corresponds to the impact of the variable v_j on the variable v_i. The stronger the impact, the heavier the edge, the darker the cell. The matrix is ordered according to the order of the hidden kernel of variables[5].

Search is initiated with singleton consistency propagation (every value of every variable is propagated (Prosser et al, 2000)) to ensure that the impacts of variables are homogenously initialized. Although the graph is almost entirely connected, the matrix-based visualization depicted in Figure 1 makes it possible to see very clearly the structure of the problem, *i.e.* the three sets of variables having strong internal links, right after this first propagation step (we use here the generic impact measure I_0).

Figure 2 depicts the impact graph after two minutes of search using `mindom` as variable selection heuristic. Impacts are not used here for

[5] We are currently working on clustering algorithms (Cleuziou et al, 2003) to discover this particular ordering from the impact graph alone.

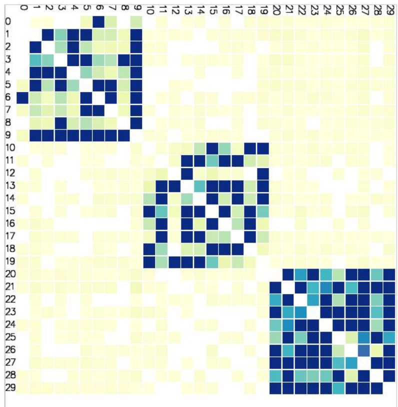

Figure 1. The impact graph GI_0 (using I_0) after the initialization phase.

search but are only maintained during search in order to generate the associated graphs. One can notice how I_0 highly concentrates on dynamically revealed structure (initial clusters are no longer visible compared to Figure 1) whereas I_3 is focused on the original static structure and interestingly forgets the weak links even after two minutes of computation (see Figure 2). The darker area for I_0 at the bottom left corner shows that the variables in the first two sets have an apparently strong influence on the variables belonging to the third set. This can be accounted for by the fact that bad decisions taken early on the variables of the first sets lead the solver into numerous try-and-fail steps on the variables of the third set hiding the existing structure in the problem. Notice that this zone would get darker if search would not have been interrupted for analysis.

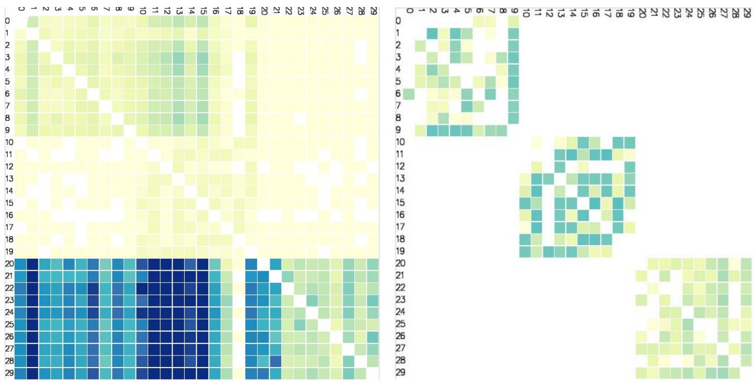

Figure 2. Impact graph GI_0 (left) and GI_3 (right) after two minutes of computation using `mindom`.

Figure 3 represents GI_1 the impact graph related to I_1. It is a normalized graph where the impact of a decision taken by the solver is divided by the number of times this decision has been taken during search so far. By doing so, we aim at refining the previous analysis by distinguishing two types of decisions: those having a great influence because they are repeated frequently, and those having a great influence because they guide the solver in some inconsistent branch of the search tree and appear in all inconsistency explanations. We can thus isolate early bad decisions that seem to involve the second set of variables.

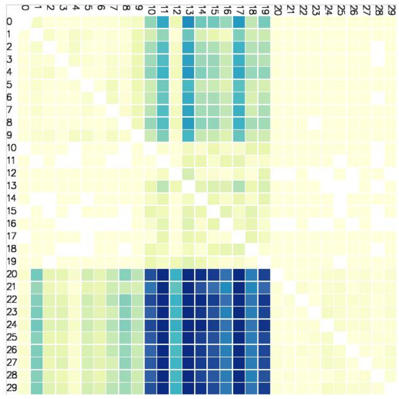

Figure 3. The impact graph GI_1 after two minutes of computation using `mindom`.

Finally, Figure 4 represents the activity within the impact graph where the effect of old decisions is gradually discarded (I_2). As expected, it appears that the solver keeps going back and forth between the first and third sets of variables, with very negligible involvement of the second set. This must be related to poor decisions taken on the variables of the second set.

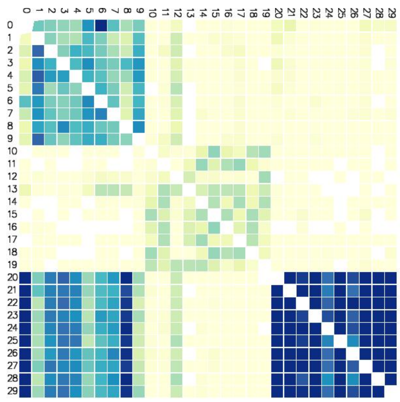

Figure 4. The impact graph GI_2 after two minutes of computation using `mindom`.

We therefore modified our search heuristic so that decisions whose apparent influence increases too much (because they appear in many explanations but do not provide any valuable pruning) are discarded as soon as possible. The problem was then solved almost instantaneously.

4. Using impacts to improve search

In this section, we illustrate how the impact measure introduced above can be used in order to improve search techniques.

4.1. Branching strategies

Classical branching strategies take into account the current domains (`mindom`) and/or the degree of the variables in the constraint network (`dom + deg` or `dom/deg`) in order to identify most constrained variables whose instantiation will simplify the problem. Impacts as presented above naturally generalize this idea.

4.1.1. *Variable-oriented impacts*

Impacts have been aggregated to express relationships between variables. In order to be used for branching, we need a problem-wide aggregation. For measures I_0 to I_2, the global impact of a decision is computed by aggregation on the whole set of variables in the problem[6]:

$$\forall \alpha \in [0,1,2], \qquad I_\alpha(x_i = a) = \sum_{x_j \in X} I_\alpha(x_i = a, x_j)$$

Upon branching, first, we choose the variable x that maximizes $\sum_{a \in D(x)} I(x = a)$ and second, for that variable, we choose the value v that minimizes $I(x = a)$ in order to allow a maximum possible future assignments ($D(x)$ is here the current domain of x).

4.1.2. *Constraint-oriented impacts*

Impacts have been defined based on decision constraints ins order to reveal relationships between variables through search. But, they can be defined in a similar way for any constraint ct following I_0:

$$I(ct) = \sum_{\{e \in E, ct \in e\}} 1/|e| \qquad (2)$$

[6] I_3 is handled differently as we need to focus on search space reduction as it is done in Section 3.3 for $I_3(x_i = a, x_j)$.

One of the best branching strategies nowadays is `dom/deg` introduced in (Bessière and Régin, 1996). `dom/deg` selects the variable that minimizes the ratio between the size of its current domain and its degree in the constraint network. The idea is to favor smaller domains and most constrained variables. It has been improved recently for binary constraints by (Boussemart, 2004) leading to the `dom/Wdeg` heuristic which increments the degree of a given constraint each time it raises a contradiction. We propose here to get a step further by replacing the degree with the impact measure of the constraints on the considered variable (this is the `dom/Ict` heuristic).

4.2. Experiments

For simplicity and clarity, we have chosen to present the results obtained for the two most significative variants: I_2 et `dom/Ict`. The framework of our experiments is the following:

- As explanations are nevertheless maintained to compute the impact measures, we can easily switch from standard chronological backtracking to back-jumping (we use `mac-cbj` (Prosser, 2000));
- We will compare our strategies to `dom/deg`, `dom/Wdeg` and I_R, the impact search based on the principles of (Refalo, 2004).I_R compares the remaining search space (taking the Cartesian product of the domains) after and before each decision in order to precisely quantify the search space reduction. The variable that maximizes the search space reduction is chosen and the value that minimizes it. In order to test the influence of the back-jumping mechanism, I_R and `dom/deg` are both tested using `mac` and `mac-cbj`. The best combination is reported;
- All ties are randomly broken;
- Finally, experiments were conducted on a Pentium 4, 3 GHz running Windows XP. We use `choco`, an open-source constraint library in Java (`choco-solver.net`).

(Refalo, 2004) mentions that initializing impact is critical and using *restart* can pay off when initialization is not able to efficiently approximate impacts. Note that impacts become more pertinent as time goes by. The two following techniques were therefore implemented:

- *restart* procedures enforce an increasing limitation on the number of nodes authorized during search. This limit is doubled at each iteration providing a complete technique. Results with *restart* with an heuristic h are denoted in the results $h + rest$. They are mentioned as soon as they outperform h alone.

- impacts are initialized using a propagation phase similar to singleton consistency (Prosser et al, 2000) (each value for each variable is propagated). The cost of this initialization is reported in the indicated times.

We considered three sets of benchmark problems:

1. The first set comes from experiments in (Refalo, 2004): a set of multiknapsack problems modelled with binary variables. Each problem is solved as a satisfaction problem (the optimal value for the original problem is set as a hard constraint). For this set a time limit of 1500s is considered. We report here an average for 30 executions of the algorithms for solving each instance (recall that ties are randomly broken).

2. Our second set is made of random structured instances made as described in Section 3.4. A $< 45, 10, 35, p_2 >$ problem is structured with three kernels of 15 variables linked with an intra-kernel tightness p_2 and an inter-kernel tightness of 3%. 100 instances are considered for each value of p_2. Average results are presented.

3. Our last set is made of real world frequency allocation problems (Cabon, 1999) coming from the FullRLFAP archive. The problem resides in finding frequencies (f_i) for different channels of communication minimizing interferences. Interferences are expressed using minimal distance constraints between frequencies of some channels ($|f_i - f_j| > E_{ij}$ or $|f_i - f_j| = E_{ij}$). We followed (Bessière et al, 2001; Boussemart, 2004) to generate hard satisfaction instances. Therefore, `scenXX-wY-fZ` will correspond to the original instance `scenXX` where constraints with a weight greater than `Y` are removed as well as the `Z` highest frequencies. Results are reported on a set of 15 hard instances identified by (Bessière et al, 2001; Boussemart, 2004).

4.3. First benchmark : multiknapsack problems

On this first benchmark (results are reported in Table I), I_R is the best strategy. We get the same results as in (Refalo, 2004) regarding I_R. The use of *restart* does not provide any improvement for any of the tested heuristics. The random tie-breaking takes a too large part for I_2 and `dom/Ict`[7]. Those two heuristics are not pertinent here as they require a too long *learning* before impacts get to stabilize and allow variable discrimination. It is quite clear for I_2 when looking at the number of nodes of the last iteration upon *restart* (*cf.* Table I). Thus, *restart* makes I_2 a viable alternative but it takes too long and the overall time necessary for solving the problem increases.

Table I. Impacts on multiknapsack problems randomly breaking ties (average results on 30 executions)

	mac dom/deg		mac I_R		mac dom/Wdeg	
	Tps (s)	Nodes	Tps (s)	Nodes	Tps (s)	Nodes
mknap1-2	0	11.2	0	24.3	0	11.9
mknap1-3	0	85.9	0	165.7	0	89.8
mknap1-4	0.3	2236.7	0.2	1149.5	0.4	2506.1
mknap1-5	3.6	27749.1	3.5	23158.1	4.7	32437.6
mknap1-6	316.8	2031108.5	201.1	1066116.4	452.9	2636561.5

	dom/Ict		I_2		I_2 + rest	
	Tps (s)	Nodes	Tps (s)	Nodes	Tps (s)	Nodes
mknap1-2	0	32.8	0	26.0	0	26.0
mknap1-3	0.1	334.5	0.1	594.3	0.1	200.8
mknap1-4	4.3	15063.8	2	7141.5	6	6770.8
mknap1-5	723	2881651.4	234	861328.5	317	446652.6
mknap1-6	> 1500		> 1500		> 1500	

4.4. Second benchmark: structured random binary problems

On this benchmark, `mac-cbj` seems critical (`mac dom/deg`, `mac` I_R or `mac` $I_R + rest$ are not able to compete) except for `dom/Wdeg` (Figures 5 and 6 report the obtained results). I_2, `dom/Wdeg` as well as `dom/Ict` are much more efficient than I_R which is better than `dom/deg`. Here, *restart* does not pay off for I_2 and `dom/Ict`. However, *restart* with I_R

[7] Results are much more better (for all heuristics) on this benchmarks when ties are broken deterministically.

is a good strategy as the initialisation phase is insufficient to get useful information.

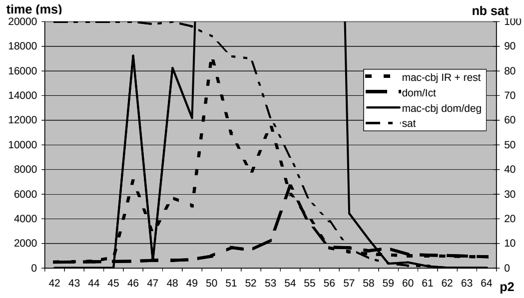

Figure 5. Average resolution time (left axis) and number of satisfiable instances (sat) (right axis) for dom/deg, I_R and dom/Ict.

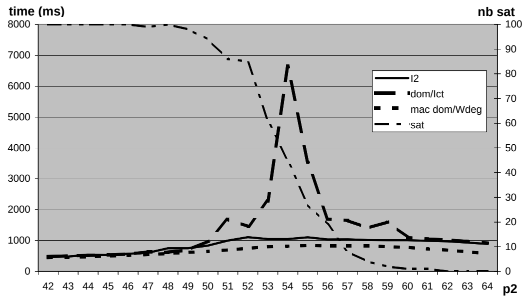

Figure 6. Average resolution time (left axis) and number of satisfiable instances (sat) (right axis) for dom/Ict, I_2 and dom/Wdeg.

The efficiency of I_2 and `dom/Wdeg` is probably due to the fact that the complexity of this benchmark does not only rely at the instance level but also because of the high degree of interaction with the search algorithm. Artificial structures are in favor of *heavy-tailed* behaviors (William et al, 2003a) which is characterized by a great variation in pure random search leading to highly difficult to identify bad initial choices. Notice that the bad behavior for I_R starts just before the phase transition.

4.5. Third benchmark: frequency allocation

On this benchmark, `mac-cbj` is crucial as shown in the first two columns in Table II (7 solved instances for `mac-cbj` as opposed to only 3 for `mac`). A way of analyzing this result is to look at the impact graph. The first matrix on Figure 7 gives the impact graph at the end of the initialization phase. One can immediately see that the constraint network is really sparse. Moreover, even after a repeated random search (for a limited number of nodes) in order to let indirect relations appear (the second matrix on Figure 7), impacts do not appear throughout the problem but stay quite localized. Thrashing commonly arises in such situations. As propagation remains quite local, a contradiction can be discovered quite late on a part of the graph after a long search on another part (because the two parts are poorly related).

Results for the four stragegies[8] are presented on Table II. We systematically use here `mac-cbj`. I_R is better that I_2 which is in turn better than `dom/deg`. The initialization phase can be quite costly (up to 40 seconds in the worst cases) but can alone prove the inconsistency of some instances. Initialization is not applied for `dom/Wdeg` as it does not need it compared to `dom/Ict`.

`dom/Wdeg` is the winning strategy on this benchmark. However, our new strategy `dom/Ict` is as efficient (in terms of solved instances – 12 out of the 15 instances considered[9]) and is only outperformed in terms of time because of its necessary initialisation step.

Table II. Impacts on frequency allocation problems

		`mac dom/deg`	`mac-cbj dom/deg`	$I_2 + rest$	$I_R + rest$	`dom/Ict`	`dom/Wdeg`
`scen11`	Time (s)	47	7.9	38	53	40	11.5
(sat)	Nodes	5863	1207	1432	3986	524	907
`scen02-f24`	Time (s)	0.8	0.1	3	3	3	0.1
(sat)	Nodes	620	104	88	90	104	95
`scen02-f25`	Time (s)	$> 1h$	3.6	4.6	3.7	8.5	1.4
(unsat)	Nodes	-	610	270	77	1252	83
`scen03-f10`	Time (s)	$> 1h$	1766	11.5	9.7	10.5	0.5
(sat)	Nodes	-	527507	1128	415	186	188
`scen03-f11`	Time (s)	$> 1h$	$> 1h$	$> 1h$	$> 1h$	17.5	19.6
(unsat)	Nodes	-	-	-	-	788	1369
`scen06-w2`	Time (s)	$> 1h$	75	14.6	13.5	15.8	1.1
(unsat)	Nodes	-	68669	0	0	0	78
`scen07-w1-f4`	Time (s)	0.2	0.2	6	5.9	6.9	0.3
(sat)	Nodes	271	202	194	191	185	207
`scen07-w1-f5`	Time (s)	$> 1h$	0	4.4	4.3	5	0.1
(unsat)	Nodes	-	26	0	0	0	29
`graph08-f10`	Time (s)	$> 1h$	$> 1h$	$> 1h$	679	19	14
(sat)	Nodes	-	-	-	200898	757	1392
`graph08-f11`	Time (s)	$> 1h$	$> 1h$	$> 1h$	174	14	3.3
(unsat)	Nodes	-	-	-	32653	25	254
`graph14-f27`	Time (s)	$> 1h$	$> 1h$	14.9	26.2	32.9	3.7
(sat)	Nodes	-	-	4886	9845	7080	1817
`graph14-f28`	Time (s)	$> 1h$	$> 1h$	$> 1h$	$> 1h$	14.3	4
(unsat)	Nodes	-	-	-	-	1377	901
nb solved		3/15	7/15	8/15	10/15	12/15	12/15

[8] I_R needs to compute the Cartesian product of domains. In order to discard integer overflowing, we based I_R on the measure of the reduced values instead of the remaining space.

[9] 3 out of the 15 instances could not be solved by none of the tested techniques.

Figure 7. The impact graph at the end of the initialisation phase and after a repeated random search of fixed size.

4.6. Impact-based strategies: first insights

Strategies I_α (α ranging from 0 to 2) are strongly related to the solver's activity during search (thus, focussing on the dynamically revealed component of the structures). Their use can be quite efficient as they reveal bad initial choices (whose influence will grow incommensurably during search without any useful pruning because the solver cannot manage to get back to them). Moreover, when applying impacts to constraints, the ones responsible for the propagation are identified. But, we think that the explanation-based impact varies too much between two different nodes in the search tree so that I_α will not make robust generic strategies alone. I_3 is clearly too costly (*wrt.* time) in its current implementation to be used as default heuristic.

5. Specific structures exploitation

Benders decomposition in OR is a technique dedicated to problems that have a static structure: a subset of variables which has a great impact on the others and for which the remainder of the problem can be decomposed in independent sub-problems. The decomposition uses a master-slave relationship between variables. The master variables are called *complicating variables* by Geoffrion (1972). Once these variables instantiated, the resulting optimization subproblem is much more simple.

Our aim here is to use this decomposition technique, which has proven quite efficient in OR, for our *hidden* structures revealed by the impact graphs we defined (we have subsets with strong intra-relations and light inter-relations). Using this technique in CP is not immediate. Usually, classical Benders cuts are limited to linear programming and are obtained by solving the dual of the subproblem. Therefore, they require that dual variables or multipliers to be defined to apply the decomposition. However, Hooker and Ottosson (2003) proposed to overcome this limit and to enlarge the classical notion of *dual* by introducing an *inference dual* available for all kinds of subproblems. They refer to a more general scheme and suggest a different way of considering duality, a Benders decomposition based on *logic*: being able to produce a proof and a sufficient set of hypothesis to justify this proof.

However this inference dual must be implemented for each class of problems to derive accurate Benders cuts (Jain and Grossmann, 2001). One way of thinking the dual is to consider it as a certificate of optimality or an *explanation* (as introduced in Section 3.1) of inconsistency in our case. Our explanation-based constraint programming framework therefore provides in a sense an implementation of the logic-based Benders decomposition in case of satisfaction problems (Cambazard et al, 2004). One can notice here as the computation of explanations is *lazy*[10], the first explanation is taken whereas several explanations exist. One cannot look for the minimal explanation for evident scalability reasons. Therefore, such an inference dual provides an arbitrary[11] 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 subproblem.

Explanation-based constraint programming as used in algorithms like `mac-dbt` (Jussien and al, 2000) or in `decision-repair` (Jussien and Lhomme, 2002) automatically focus on the master problem of such a decomposition but may revert to a more conventional behavior (similar to `mac`) when independence is not properly identified. The next step would be here to use the structure exhibited from the impact graphs presented above in order to apply a Benders decomposition scheme in a second phase of resolution. The identification of sub-structures once the master instantiated could guide the generation of cuts for the master to gather as much information as possible where lies the real combinatorics of the problem.

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

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

6. Conclusion

In this paper, we introduced several indicators useful for both identification and use while searching of key structures at the heart of combinatorial problems. We focused our study on the relationship between variables and gave new perspectives on the design of generic search heuristics for constraint programming as well as search algorithms. We believe that the presence of *backdoors* or subset of variables exhibiting a strong impact over the whole problem could be explicitly used by *ad hoc* decomposition or relaxation strategies inspired from Operation Research. A concrete example is Benders decomposition and its generic extension based on logic. It is indeed exactly a *backdoors* technique and could be applied in Constraint Programming as a *nogood* learning strategy.

References

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.

J. F. Benders. Partitionning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

C. Bessière, A. Chmeiss, and L. Sais. Neighborhood-based variable ordering heuristics for the constraint satisfaction problem. In *Proceedings CP'01*, pages 565–569, 2001.

C. Bessiere and J.-C. Régin. MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems In *Proceeding CP'96*, pages 61-75, 1996.

F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings ECAI'04*, pages 482–486, 2004.

B. Cabon, S. de Givry, L. Lobjois,T. Schiex,J.P. Warners", Radio Link Frequency Assignment. In journal *Constraints*, volume 4, pages 79-89, 1999.

H. Cambazard, P.-E. Hladik, A.-M. Déplanche, N. Jussien, and Y. Trinquet. Decomposition and learning for a real time task allocation problem. In *Proceedings CP 2004*, pages 153–167, 2004.

H. Cambazard and N. Jussien. Integrating Benders decomposition within Constraint Programming In *Proceedings CP 2005*, short paper, pages 752–756, 2005.

G. Cleuziou, L. Martin, and C. Vrain. Disjunctive learning with a soft-clustering method. In *ILP'03:13th International Conference on Inductive Logic Programming*, pages 75–92. LNCS, September 2003.

A. M. Geoffrion. Generalized Benders Decomposition. *Journal of Optimization Theory And Practice*, Vol. 10, No. 4, pages 237–260, 1972.

M. Ghoniem, N. Jussien, and J.-D. Fekete. VISEXP: visualizing constraint solver dynamics using explanations. In *Proceedings FLAIRS'04*, Miami, Florida, USA, May 2004.

C. P. Gomes, B. Selman, and N. Crato. Heavy-tailed distributions in combinatorial search. In *Proceeding CP 1997*, pages 121–135, Linz, Austria, 1997.

R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(9):263–313, 1980.

J. N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96:33–60, 2003.

V. 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.

N. 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.

N. Jussien and V. 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.

N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings CP 2000*, pages 249–261, Singapore, 2000. Springer-Verlag.

N. Jussien and O. Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, July 2002.

R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman and L. Troyanski. Determining computational complexity for characteristic 'phase transitions'. In *Nature*,volume 400, pages 133–137, 1999.

P. Prosser MAC-CBJ: maintaining arc-consistency with conflict-directed back-jumping. Research report 95/177, Department of Computer Science – University of Strathclyde, 2005.

P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In R. Dechter, editor, *Proceedings CP 2000*, pages 353–368, Singapore, 2000.

P. Refalo. Impact-based search strategies for constraint programming. In *Proceedings CP 2004*, pages 556–571, Toronto, Canada, 2004.

J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *AAAI 94, Twelth National Conference on Artificial Intelligence*, pages 362–367, Seattle, Washington, 1994.

R. Williams, C. P. Gomes, and B. Selman. On the connections between backdoors and heavy-tails on combinatorial search. In *the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2003.

R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings International Joint Conference in Artificial Intelligence*, 2003.