

Identifying and exploiting problem structures using explanation-based constraint programming

Hadrien Cambazard and 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. 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 dynamic and static structure of a problem instance. We define several impact graphs to be used to design generic search guiding techniques and to identify hidden structures of instances. Finally, 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 techniques for solving combinatorial problems seems like the Holy Grail for both OR and CP communities. Several tracks are now explored: dynamically analyzing and adapting the way the solver actually solves a combinatorial problem, identifying specific structures in a given instance in order to speed up search, etc. The key point is to be able to identify, understand and use the intimate structure of a given combinatorial problem instance [7, 17, 18].

Refalo [16] recently defined impact-based solving strategies for constraint programming that dynamically use the structure of a solved problem. In this paper, we attempt to investigate the relationships between the variables of the problem. We intend to identify, differentiate and use both dynamic (created by the search algorithm) and static (relative to the instance) structures of the problem being solved. We focus on structures intended as subsets of variables that play a specific role within the problem. We define to this end several fine grained impact measures and induced impact graphs between variables to:

- identify hidden structures in problems;

- design generic search guiding techniques;
- pave the way of possible use of the impact analysis into decomposition based methods such as, for example, 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.

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 dynamic 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 The idea of exploiting problem structures within search strategies

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

A problem is more generally said to be *structured* if its components (variables¹ 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 difficulty in identifying structure in problems is that this structure is not always statically (at the instance level, before solving) present. 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 *dynamic structure*. It is related to bad initial choices as well as new relationships due to the addition of constraints during the search. This does not make things easy when willing understand the complexity of a problem and use this information to speed up search techniques.

Backdoors, recently introduced in [18], are an interesting concept to characterize hidden structure in problem instances. They are informally defined as subsets of variables that encapsulate the whole combinatorics of a given instance of a problem: once this core part completely instantiated, the remaining subproblem can be solved very efficiently. Numerous search strategies are based, knowingly or not, upon this principle. The following two were the most influential for our work:

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

- Branching heuristics in CP attempt to early guide the search towards the backdoors 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 lead the possibly smallest search space and that raises contradictions early as possible. This principle (often referred to the *first fail* principle [8]) is often implemented by taking the current domain and degree of constrainedness (see [3] for variants) of the variables into account. More recently, [16] proposed to characterize the impact of a choice and a variable by looking at the search space reduction caused by this choice in average (another way of identifying a *backdoor*) and used this information as a guiding strategy.
- Benders decomposition [2] 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, it 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 who 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 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 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 *backdoors* because, thanks to condition 1, once completely instantiated the remaining problem can be solved efficiently. Moreover, if the remaining subproblem can be actually solved polynomially (this is referred as *strong* backdoors), a powerful cut based on the minimal conflict can often be computed.

For the latter technique, 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 *all-different* constraint to tighten propagation on the problem. But, such an analysis only provides information on visible static structures. Nevertheless, hidden structure and dynamic one seems 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. We believe that the propagation performed by the solver during search provides information

² This cut is often referred as the Benders cut.

that should lead to identify those hidden structures. One way of exploiting that information is to use explanations.

3 Identifying problem structure using explanations

Refalo [16] 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 claim here that we can go a step further by analyzing where this propagation 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 being to identify variables that maximally constrain the problem as well as subsets of variables that have strong relationships and strong impact upon the whole problem (namely a *backdoors*). 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.

Such information relies on the concept of explanation for CP [11].

3.1 Explanations for constraint programming

Explanations have been initially introduced to improve backtracking-based algorithms but have been recently used for many purposes including dynamic constraint satisfaction problems and user interaction.

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 original constraints of the problem) and a set of decisions dc_1, \dots, dc_n taken during search. An explanation of the removal of value a from variable v will be written: $C' \wedge dc_1 \wedge dc_2 \wedge \dots \wedge dc_n \Rightarrow v \neq a$.*

As the constraint solver always know (although may be not explicitly) *why* it removes a value from the domain of a variable, explanations can be computed within the solver³ [12]. Thus, explanations computed by the solver account for the underlying logical chain of consecutive inferences made by the solver during propagation. In a way, explanations provide an accurate trace of the behavior

³ Notice that when a domain is emptied (*i.e.* a contradiction is identified), an explanation for that situation is computed by uniting each explanation of each removal of value of the variable concerned.

of the solver as all operations are explained. In the following, E_i^{val} will denote the set of all explanations computed from the start of the search for all different removals of the value val of the variable i that have occurred throughout search.

3.2 Characterizing impact

The impact of a decision $x_i = a$ can be expressed, according to the first fail principle, through the reduction of the search space implied in average by this decision. 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.

We introduce now our new measures whose aim is to characterize the impact of a decision not only based on the immediate search space reduction. We denote $I_\alpha(x_i = a, x_j, val)$ the impact of taking decision $x_i = a$ on the value val of a variable x_j . α is an index used to distinguish our different measures.

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. Therefore, small explanations reveal strong relationships.

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

From this basic measure, we introduced different impact measures based on the solver activity and the computation of explanations (measures I_1 and I_2) in order to exhibit dynamic structures. We also designed a search space reduction based measure (I_3) in order to capture static structures and to help guiding search. As search obviously direct propagation (and vice-versa), it seemed quite natural to normalize this basic measure according to search.

- The impact is here normalized according to the number of times a decision $x_i = a$ is taken during search: $|x_i = a|$. We simply intend here to distinguish frequent decisions (*i.e.* most likely recent ones) and hardly reconsidered ones (*i.e.* most likely quite old ones):

$$I_1(x_i = a, x_j, val) = \frac{\sum_{\{e \in E_j^{val}, x_i = a \in e\}} \frac{1}{|e|}}{|x_i = a|}$$

- Another way to normalize is to consider the $age^4 a_e^d$ of a decision d when computing an explanation e with the aim of decreasing the impact of old decisions. This leads to:

⁴ The distance in the search tree from the decision to the resulting removal.

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

- The computation of impacts is spread within the whole resolution process as it is done during explanation computations. It is a quite different approach to [16] which analyses each decision separately to get its instantaneous impact. I_3 tries to identify recurrent search space reduction associated to a decision:

$$I_3(x_i = a, x_j, val) = \frac{\sum_{e \in E_j^{val}, x_i = a \in e} \frac{1}{|e|}}{|\{x_i = a \text{ active} \wedge val \in Dom(x_j)\}|}$$

$I_3(x_i = a, x_j, val)$ can be considered as the probability that the value val of x_j will be pruned if the decision $x_i = a$ is taken. It therefore considers the number of time a removal could have been done and the number of time it has been effectively done. 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.

3.3 From fine-grained impact measures to relations between variables

From the previous definitions, we can introduce different directed weighted graphs of impacts $GI(V, E)$ with weight $I(x, y)$ for any couple $(x, y) \in E = V \times V$. In order to define those weights, fine-grained impact measure introduced above are aggregated in the following way:

$$I(x_i = a, x_j) = \sum_{val \in D(x_j)} I(x_i = a, x_j, val)$$

where $I(x_i = a, x_j, val)$ can be replaced by any of the 4 measures introduced before ($\{I_\alpha \mid \alpha \in [0, 1, 2, 3]\}$).

We have a special case for I_3 , as it intends to relate the impact to the domain reduction generated by a variable over another. Relating a variable and a decision is therefore normalized considering the domain initial size of the variable:

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

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 $x_i = a$.

The weight of an edge can now be computed in the following way :

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

3.4 Overall impact of a given decision

For measures I_1 and I_2 , the overall impact of a decision is computed by accumulating impacts over variables of the problem:

$$I(x_i = a) = \sum_{x_j \in V} I(x_i = a, x_j)$$

As for measure I_3 , focus is made on the average search space reduction. The current size P of the search space is the Cartesian product of the current domains of variables. Therefore, the overall impact of a decision for the whole problem is expressed through its effective search space reduction by considering the probable remaining space after the decision $((P_{before} - P_{after})/P_{before})$:

$$I(x_i = a) = (P - \prod_{x_j \in V} \sum_{val \in D(x_j)} (1 - I_3(x_i = a, x_j, val)))/P$$

3.5 An illustrative case

We take here a particular instance⁵ of the benchmark problems introduced later in Section 4 in order to illustrate what kind of structures are isolated by our impact measures. Moreover, we will describe how the retrieved information may be used at the user level to investigate problems and instances.

We therefore consider a random binary problem in which a structure is inserted by increasing the tightness of some constraints in order to design several subsets of variables with strong relationships. Random instances are 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. 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` [8] classical variable selection heuristic. Using the different measures of impact introduced above, we would like to illustrate how several questions may be addressed when facing a problem:

- 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 ?

⁵ One of the random generated problem with a given set of parameters.

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 [6]: 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 variables⁶.

Notice that we start search by applying a kind of singleton consistency propagation (every value of every variable is propagated [15]) 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).

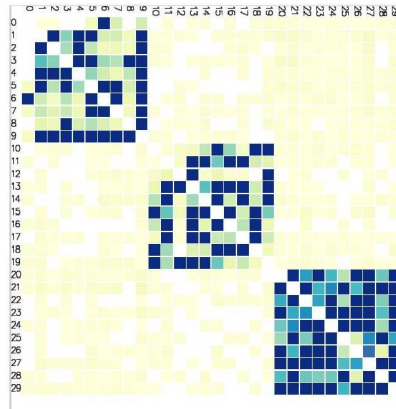


Fig. 1. The representation of the impact graph of variables at the end of the initialization phase using I_0 as measure of impact

Figure 2 depicts the impact graph after two minutes of search using `mindom` as variable selection heuristic (using both I_0 and I_3). One can notice how I_0 highly concentrates on dynamic 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. 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.

Figure 3 represents a normalized representation of the same graph where the influence of a decision taken by the solver is divided by the number of times

⁶ We are currently working on clustering algorithms [5] to discover this particular ordering from the impact graph alone.

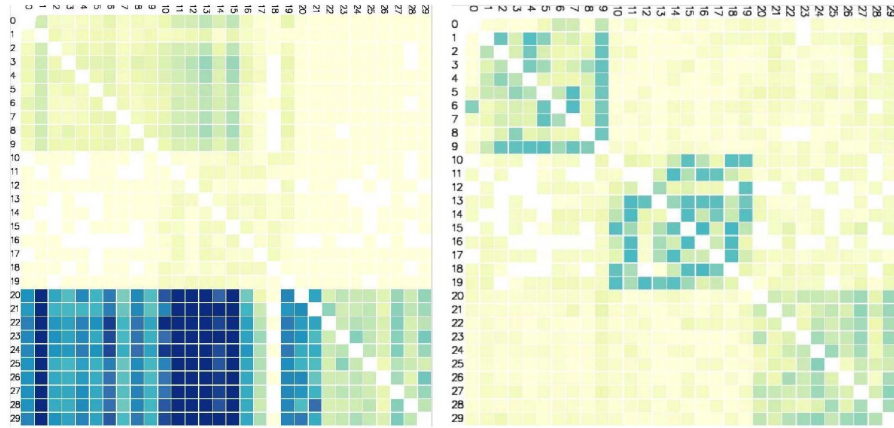


Fig. 2. The impact graph using I_0 (on the left) and I_3 (on the right) after two minutes of computation using the `mindom` heuristic.

this decision occurred during the resolution (measure I_1). 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.

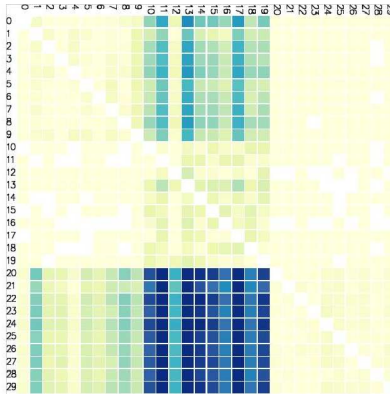


Fig. 3. A representation of the impact graph normalized according to the number of times a decision is taken (I_1).

Finally, Figure 4 represents the activity within the impact graph where the effect of old decisions is gradually discarded. As expected, it appears that the solver keeps going back and forth between the first and third set 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.

In order to further confirm this interpretation, we adapted our search heuristic so that it takes into account the impact of variables during the resolution and

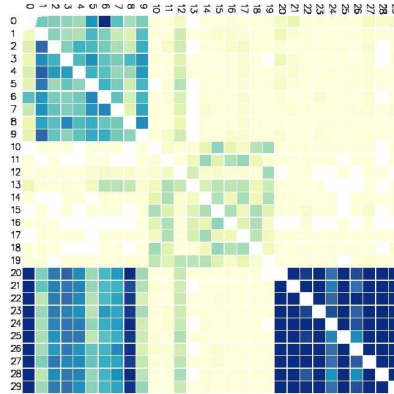


Fig. 4. A representation of the impact graph normalized according to the age of decisions (I_2)

undoes immediately decisions whose influence increase outstandingly (because they appear in many explanations but do not provide any valuable pruning). 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. We use the impact measure in the branching heuristics within a tree search: 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). Ties are randomly broken. As said earlier, impacts are initialized through the use of a singleton consistency-like propagation.

Our experiments were conducted on a Pentium 4, 3 GigaHz, running Windows XP. Our constraint solver is the most recent Java version of `choco (choco.sf.net)`. Notice that as we are using explanations, our tree search is not limited to standard backtracking but we actually use the `mac-cbj` algorithm (getting higher in the search tree if it is possible upon encountering a contradiction). In practice, the behavior is very close to `mac` and behaves as it was only merely maintaining explanations. We considered three sets of benchmark problems:

1. The first set comes from experiments in [16]: a set of multiknapsack problems modelled with binary variables. For this set a time limit of 1500s is considered. We focused here on the number of developed nodes⁷ as it is directly related the relevance of the measure. Moreover, as randomness is introduced in the problem solving when breaking ties, we report an average over 10 executions.

⁷ In the presented results, when a restart technique is used, only the number of nodes of the last execution are reported whereas the overall time is indicated.

2. Our second set consists on random binary problems generated following the classical B model (see Section 3.5) with the following parameters: $\langle 50, 10, 30, p_2 \rangle$. We considered here a time limit of 120s. We focus on the number of unsolved instances within the time limit for each value of p_2 .
3. Our final set is made of random structured instances made as described in Section 3.5. A problem $\langle 45, 10, 35, p_2 \rangle$ is structured with three kernels of 15 variables linked with an intra-kernel tightness p_2 and an inter-kernel tightness of 3%.

As for the impact measure, we compared three measures ($\{I_\alpha \mid \alpha \in [1, 2, 3]\}$) and our implementation of the measure introduced in [16] (denoted I_{ref}). As the measure completely specifies the search used, we will refer in the following to the I_α and I_{ref} strategies in the following.

4.1 First benchmark: multiknapsack problems

On this first benchmark (whose results are reported on Table 1), I_{ref} appears to be the best search strategy. The use of explanations seems to provide good information but it is a long term learning (it requires a restart policy) and is much more costly (in time) so that it cannot solve the instance `mknap1-6`. I_3 is obviously too costly on this problem where near one million nodes need to be explored. The number of nodes of `mindom` is given here as a reference.

Table 1. Impacts on multiknapsack problems

	<code>mindom</code>	I_{ref}		I_3		I_3 +restart	
	Nodes	Time	Nodes	Time	Nodes	Times	Nodes
mknap1-2	38	0	25.9	0	23.1	0	23.1
mknap1-3	385	0.1	188.7	0.3	354.1	0.3	255.2
mknap1-4	16947	0.7	982.7	4.2	2754	3.2	979.5
mknap1-5	99003	11.2	21439.6	229.1	110666.4	112.8	20237.4
mknap1-6	21532776	425.7	612068	> 1500		> 1500	

I_1 and I_2 are not accurate on these instances and maybe need a fine restart policy as they attempt to detect irrelevant first choices. As mentioned by Refalo, the use of restart only increases the overall computation time for I_{ref} but seems to be important for I_3 . I_3 is indeed a fine-grained measure that maybe need more time to become accurate for the search.

4.2 Second benchmark: random binary problems

On this unstructured benchmark, the size of the domains (integer variables instead of binary ones) gives to `mindom` better chances to make good choices. the results (depicted in Figure 2) are not in favor of impact measures alone. However, their combination with `mindom` as a way of breaking equalities is much

more powerful and allow to solve around 93 % of instances over the whole phase transition against 79 % for `mindom` alone. This combination avoids bad choices for I_{ref} which becomes the best technique whereas it was the worst one alone (I_1 could solve 17 % more instances than I_{ref}). The use of restart generally increases the overall computation time.

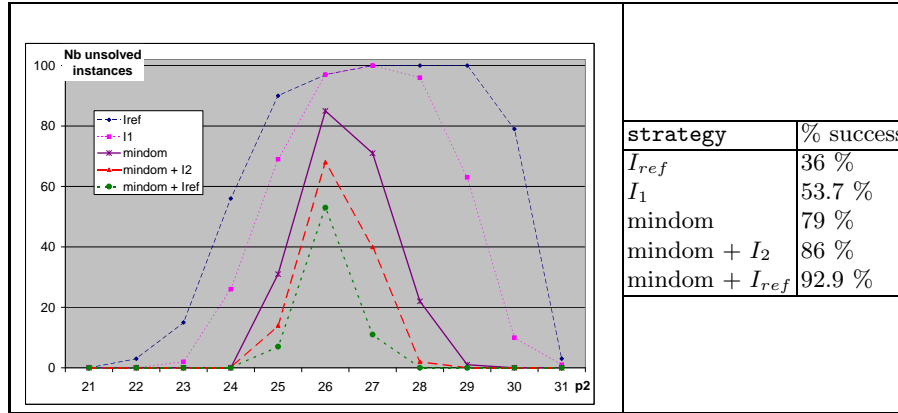


Table 2. The number of unsolved instances (left) for each impact strategy on $< 50, 10, 30, p_2 >$ and the percentage of successfully solved instances (right).

4.3 Third benchmark: structured random binary problems

On this set of problems, the I_{ref} strategy seems to experience difficulties even compared to the classical `mindom` heuristic. Figure 5 reports the number of instances that were not successfully solved within the time limit of 120s. As restart does not help the I_{ref} strategy again on this problem but is effective for I_3 , only the best results (*i.e.* with or without restart) of each technique are indicated. The more impressive results here are obtained again by focusing on the dynamic component of the inherent structure of the instances (*i.e.* using strategies I_1 and I_2). That is the only way that all the instances could be successfully solved. Notice that I_3 gives better results than I_{ref} despite its high cost.

The success of I_1 and I_2 may be due to the fact that the complexity of this benchmark does not reside purely in the instances but is more due to the level of the interaction with the search algorithm. The presence of such artificial structures favors from our point of view a kind of heavy tailed behavior and makes initial choices more critical. It can indeed be noticed on Figure 6 that I_1 is sometimes subject to *bad behavior* which does not only appear at the transition phase. The same phenomenon (on a larger scale) may be the cause of the poor performance of I_{ref} .

4.4 Impact-based heuristics: first insights

I_1 and I_2 are strongly based on the solver activity during search (thus focusing on the dynamic component of the instance structure). It generally pays off

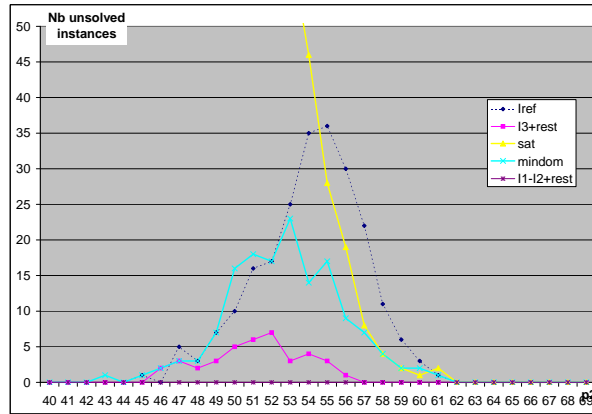


Fig. 5. Number of unsolved instances for different impact measures on random structured binary CSP only with the number of feasible instances (sat).

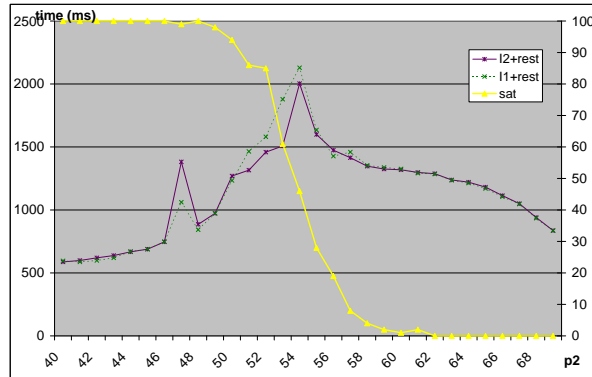


Fig. 6. Resolution time for I_1 and I_2 on random structured binary CSP with the number of feasible instances (sat).

using them on problems because (and that may be explains the relatively poor performance of I_{ref} in our benchmarks) they are able to detect past bad choices (those whose influence increases outstandingly throughout search without leading to solutions) do be undone.

I_3 is too costly (regarding time consumption) at the present time to be used as a default heuristic but some interesting compromise between I_{ref} and I_3 may be designed: taking advantage of the general robustness of I_{ref} while at the same time avoiding heavy-tailed behavior due to bad initial choices.

5 Perspective: automated logic based Benders decomposition in constraint programming

We are interested in Benders decomposition as it is intended for problems with a specific structure and specially, a master-slave relationship between variables.

For us, the master set of variables could be restricted to a subset of variables exhibiting a strong overall impact over the whole problem.

Usually, classical Benders cuts are limited to linear programming and are obtained by solving the dual of the subproblem⁸ and therefore requires that dual variables or multipliers to be defined to apply the decomposition. However, [9] proposes to overcome this limit and to enlarge the classical notion of *dual* by introducing an *inference dual* available for all kinds of subproblems. He refers to a more general scheme and suggests a different way of considering duality: a Benders decomposition based on *logic*.

However this inference dual must be implemented for each class of problems to derive accurate Benders cuts [10, 4]. 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 [4]. One can notice here as the computation of explanations is *lazy*⁹, 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 arbitrary¹⁰ 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` [13] or in `decision-repair` [14] kind of automatically focus on the master problem of such a decomposition but may be trapped by bad decisions and revert to a more conventional behaviour. 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 substructures 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.

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

⁸ Referring to linear programming duality.

⁹ Not all possible explanations are computed when removing a value. Only the one corresponding to the solver actual reasoning is kept.

¹⁰ This can also be accounted for linear duality where any dual solution is a bound for the primal problem.

Research. A concrete example is the 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

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. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.
3. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings ECAI'04*, pages 482–486, 2004.
4. 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.
5. 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.
6. Mohammad Ghoniem, Narendra Jussien, and Jean-Daniel Fekete. VISEXP: visualizing constraint solver dynamics using explanations. In *Proceedings FLAIRS'04*, Miami, Florida, USA, May 2004.
7. Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-tailed distributions in combinatorial search. In *Proceeding CP 1997*, pages 121–135, Linz, Austria, 1997.
8. R. Haralick and G. Elliot. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(9):263–313, 1980.
9. J.N. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96:33–60, 2003.
10. 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.
11. 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.
12. 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.
13. Narendra Jussien, Romuald Debruyne, and Patrice Boizumault. Maintaining arc-consistency within dynamic backtracking. In *Proceedings CP 2000*, pages 249–261, Singapore, 2000. Springer-Verlag.
14. Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. *Artificial Intelligence*, 139(1):21–45, July 2002.
15. P. Prosser, K. Stergiou, and T. Walsh. Singleton consistencies. In R. Dechter, editor, *Proceedings CP 2000*, pages 353–368, Singapore, 2000.
16. Philippe Refalo. Impact-based search strategies for constraint programming. In *Proceedings CP 2004*, pages 556–571, Toronto, Canada, 2004.
17. Ryan Williams, Carla Gomes, and Bart Selman. On the connections between backdoors and heavy-tails on combinatorial search. In *In the International Conference on Theory and Applications of Satisfiability Testing (SAT)*, 2003.
18. Ryan Williams, Carla P. Gomes, and Bart Selman. Backdoors to typical case complexity. In *Proceedings IJCAI 2003*, 2003.