A generic framework for solving CSPs integrating decomposition methods

L. Blet¹, S. N. Ndiaye², and C. Solnon¹,³

¹ Université de Lyon - LIRIS
² Université Lyon 1, LIRIS, UMR5205, F-69622 France
³ INSA-Lyon, LIRIS, UMR5205, F-69621, France

{loic.blet, samba-ndojh.ndiaye, christine.solnon}@liris.cnrs.fr

Abstract. Many real-world constraint satisfaction problems are structured, i.e., constraints are not uniformly distributed among the set of variables. This structure may be used to improve the solution process of these problems. In particular, backtracking with tree decomposition (BTD) exploits the structure to define variable ordering heuristics and to learn structural goods and nogoods which are used to avoid redundant explorations. BTD is based on a chronological backtracking search. Our goal in this paper is to investigate the interest of exploiting structure when using other approaches for exploring the search space: other complete search approaches, such as conflict directed backjumping (CBJ), but also incomplete approaches, such as decision repair (DR). To this aim, we describe a generic framework for solving CSPs, which is an extension of the framework proposed by Pralet and Verfaillie in [PV04]. Using our new generic framework, we reformulate some existing search procedures, including BTD. We also describe new search procedures which combine structural (no)goods with CBJ and DR. This generic framework allows us to experimentally evaluate the interest of exploiting the structure for different kinds of search procedures.

Keywords: Constraint satisfaction, Constraint graph, Tree-decomposition, Complete search, Local search

1 Introduction

Many real-world constraint satisfaction problems (CSPs) are structured, i.e., constraints are not uniformly distributed among variables. This structure may be used to improve the solution process of these problems. In particular, BTD [JT03] exploits the structure to define variable ordering heuristics and to collect structural (no)goods which are used to avoid redundant explorations. Experimental results have brought to the fore that these (no)goods can significantly speed up the solution process of structured instances.

BTD is based on a chronological backtracking (CB) search: it exhaustively explores the search tree by performing a depth first search until either a solution is found or inconsistency is proven. Our goal in this paper is to investigate the interest of exploiting structure when using other approaches for exploring the search space: other complete search approaches, such as conflict directed backjumping (CBJ) [Pro95], but also incomplete approaches, such as decision repair (DR) [JL02]. To this aim, we describe a
generic framework for solving CSPs, which is an extension of the framework proposed by Pralet and Verfaillie in [PV04]: the framework of Pralet and Verfaillie may be instantiated in different search procedures (such as, CB, CBJ, or DR); we extend it by adding structural (no)goods to it. Using our new generic framework, we reformulate some existing search procedures, including BTD. We also describe new search procedures which combine structural (no) goods with CBJ and DR. This generic framework allows us to experimentally evaluate the interest of exploiting the structure for different kinds of search procedures.

Section 2 introduces tree decompositions and structural (no)goods, as well as the idea of the algorithm BTD. Section 3 describes our generic framework. Section 4 presents our first experimental results. Section 5 discusses further work.

2 Backtracking with Tree Decomposition

A tree decomposition [RS86] of a graph \( G = (X, E) \) is a tree \( T = (V, F) \) so that:

- each vertex of \( v_i \in V \) is called a cluster and is a set of vertices of \( G \), i.e. \( v_i \subseteq X \);
- each vertex of \( G \) belongs to at least one cluster of \( T \), i.e. \( \forall x_i \in X, \exists v_j \in V, x_i \in v_j \);
- for each edge of \( G \), there exists one cluster of \( T \) which contains both endpoints of \( e \), i.e. \( \forall (x_i, x_j) \in E, \exists v_k \in V, \{x_i, x_j\} \subseteq v_k \);
- for each vertex \( x_i \) of \( G \), the set of clusters of \( T \) which contains \( x_i \) (i.e. \( \{v_j \in V, x_i \in v_j\} \)) induces a connected subgraph of \( T \).

Given a tree decomposition \( T \) and a cluster \( c_i \), we note \( \text{root}_T \) the root cluster of \( T \), \( \text{father}_T(c_i) \) the cluster which is the father of \( c_i \) in \( T \) (with \( c_i \neq \text{root}_T \)), and \( \text{ancestors}_T(c_i) \) the set of clusters which are ancestors of \( c_i \) in \( T \). Figure 1 displays a CSP, its constraint graph and a tree decomposition for this graph.

The width of a decomposition \( T \) is the size of its largest cluster minus one. For example, the width of the tree in Fig. 1(c) is 2. A tree decomposition is optimal if its width is the smallest among all possible decompositions of \( G \).

In BTD [JT03], the tree decomposition \( T \) of the constraint graph is used to define a variable ordering heuristic: all variables in a same cluster are assigned before assigning variables of its children. (No)goods are used to deduce that a subproblem rooted in a given cluster \( c_i \) is already solved (or already proven inconsistent) with respect to a current assignment \( A \) and a set of (no)goods \( (N)G \). The subproblem rooted in a cluster \( c_i \) is the CSP restricted to variables of \( c_i \) and of all clusters \( c_j \) such that \( c_i \in \text{ancestors}_T(c_j) \). This subproblem is already solved (resp. proven inconsistent) with respect to a current assignment \( A \) and a set of goods \( G \) (resp. nogoods \( NG \)) if there exists a good \( g \in G \) such that \( g = A_{\{c_i \cap \text{father}_T(c_i)\}} \) (resp. a nogood \( ng \in NG \) such that \( ng = A_{\{c_i \cap \text{father}_T(c_i)\}} \)).

For example, in Fig. 1, the subproblem rooted in \( \{B, C, F\} \) is the CSP restricted to \( \{B, C, F, G, H, I\} \). This subproblem is solved with respect to the assignment \( \{B = 1, C = 2, F = 3\} \) and the set of goods \( \{\{B = 1\}{\{B,G,H\}},{F = 3}\}{\{F,I\}} \), whereas it is inconsistent with respect to the assignment \( \{B = 2, C = 3, F = 1\} \) and the set of nogood \( \{F = 1\}{\{F,I\}} \).
- $X = \{ A, \ldots, I \}$
- $\forall x \in X, D(x) = \{1, 2, 3\}$
- $C = \{ A > D, D \neq E, A \geq E, A < B, A > C, G \neq B, G \leq H, \ldots \}$

Fig. 1. (a) A CSP $P = (X, D, C)$. (b) Its constraint graph $G_P$. (c) A tree decomposition of $G_P$.

The initial problem, which is rooted in $\text{root}_T$, is solved with respect to a current assignment $A$ and a set of goods $G$ if $\text{root}_T \subseteq \text{var}(A)$ and every subproblem rooted in a child of $\text{root}_T$ is solved with respect to $A$ and $G$.

3 A Generic Search Framework

Pralet and Verfaillie [PV04] have proposed a generic framework for defining a search. This search is parametrised by (i) filtering procedures which specify how domains are filtered after each step of the search, (ii) an assignment extension procedure, which specifies how to extend the current assignment, and (iii) a repair procedure which defines how to unassign some variables in the current assignment. This generic framework may be instantiated in various existing search procedures such as CB, CBJ or DR.

In this section, we extend the generic framework of [PV04] in order to allow its instantiation to structural approaches. To this aim, we introduce three sets:

- a set $E$ of explanations which are recorded when filtering domains and used to restore domains when unassigning variables, to identify conflicts when backjumping, or to prove inconsistency;
- a set $G$ of structural goods which may be recorded and used when extending assignments to avoid redundant explorations;
- a set $NG$ of nogoods which may be recorded and used when unassigning variables to avoid redundant explorations.

Note that explanations may be removed from $E$ (once they are no longer relevant) whereas (no)goods are always relevant during the whole search, so that they are never removed from $(N)G$\(^1\).

This generic framework is described in Algo.1 and is parametrised by 4 functions:

- $\text{filter}(P,A,E,NG)$ ensures some local consistency (such as FC or AC) with respect to the current assignment $A$ and returns false if some inconsistency has been detected; true otherwise.

\(^1\) We may limit the size of $G$ and $NG$. In this case we may have to remove (no)goods from $(N)G$ when the size limit is reached.
Algorithm 1: Generic Framework

Input: a CSP $P$ and an initial assignment $A$
Output: Returns true if a solution is found; false if inconsistency is proven; ? otherwise

1. Let $E$, $G$ and $NG$ respectively be empty sets of explanations, goods and nogoods
2. repeat
3. if $\text{filter}(P, A, E, NG)$ then
4. \hspace{1em} $\text{solved} \leftarrow \text{extend}(P, A, G)$
5. \hspace{1em} if $\text{solved}$ then return true
6. else
7. \hspace{1em} $\text{inconsistent} \leftarrow \text{unassign}(P, A, E, NG)$
8. \hspace{1em} if $\text{inconsistent}$ then return false
9. until $\text{Stop}()$
10. return ?

- $\text{unassign}(P, A, E, NG)$ returns true if no more variable can be unassigned (and thus inconsistency has been proven); and false otherwise. In this latter case, it unassigns some assigned variables.
- $\text{extend}(P, A, G)$ returns true if $A$ is a solution and false otherwise. In this latter case, it extends $A$ by adding a new variable/value couple to it.
- $\text{stop}()$ returns true if a stopping criterion has been met. We consider only one instantiation, which returns true when some given CPU time limit has been reached.

3.1 algorithms

We can now combine instantiations of $\text{extend}$, $\text{unassign}$ and $\text{filter}$ to obtain different search strategies. Note that when using the tree decomposition you have to do a depth first search in the tree of clusters, as in BTD.

Some instantiation captures well-known search strategies such as:

- the classical Chronological Backtracking;
- Conflict-directed BackJumping [Pro95];
- Decision-Repair as described in [PV04];
- Backtracking with Tree Decomposition [JT03].

Some of these instantiations correspond to new search strategies. In particular, we define two new search strategies which exploit tree decompositions:

- CBJ-TD performs Conflict-directed BackJumping with Tree Decomposition;
- DR-TD performs Decision-Repair with Tree Decomposition.

Complexity: CB, CBJ, BTD and CBJ-TD are complete approaches. The time complexities of CB and CBJ are exponential in the number of variables, whereas the time complexities of BTD and CBJ-TD are exponential in the width of the tree decomposition (i.e. the size of the biggest cluster in the decomposition).

DR and DR-TD are incomplete approaches. Even if they may prove some inconsistencies, there is no guarantee for the execution to end. Thus their complexities are bounded by the function $\text{stop}()$. 
4 First Results

We considered all the binary instances of the benchmark available at www.cril.univ-artois.fr/lecoutre/benchmarks.html, which contains instances of the 2008 CSP competition as well as other new instances. By lack of space, we decide to not report here the tables of results but to discuss those. We compute a tree-decomposition for each instance by using the graph triangulation algorithm Minimum Fill-in which is known to be among the best methods for this purpose. The quality of this decomposition is considered good enough if its width is at most equal to half of the number of variables in the CSP (beyond, using tree-decomposition method does not derive enough benefit).

First, we observe that most instances of the benchmark are not structured. Only 5% of the binary instances fulfilled the condition: unsurprisingly, they are real-world instances (the REAL class in the benchmark) and graph coloring ones. We run the methods defined in the previous section on the instances using forward checking as filtering technique. Using tree-decomposition improves significantly the results on coloring graph instances. It allows to solve within 1 second 4 instances otherwise unsolved within 1 hour by CB and DR. CBJ-TD succeeds to solve within 3 minutes 1 instance unsolved by CBJ within 1 hour. But, tree-decomposition based method does not behave as well on instances in the REAL class. Except a quarter of instances, using tree-decomposition surprisingly degrades the performances. We try to analyze more precisely the features of the different problems and we observe that graph coloring instances are often very hard meaning that it is necessary to explore large parts of the search space to solve them leading to many redundancies. Whereas, the instances in the REAL class, despite their great size, are often easily solved with a good variable ordering (within 1 second by CB). Since, the major effect of exploiting the structure of a problem lies in reducing the number of redundancies in the resolution, it seems obvious that it performs better on hard instances. Actually, the solving is based on a variable ordering designed to learn a maximum number of (no)goods. But, the time elapsed during this learning phase, is balanced by avoiding a lot of redundancies thereafter on hard instances. This is not the case on easy ones where a more straightforward variable ordering is sufficient to solve the problem with a limited number of redundancies. Thereby, a crucial trade-off between the exploiting of the structure and the freedom given to variable heuristic must be accomplished.

Moreover, it is clear that the exploitation of the structure has more impact on CB than on CBJ. Indeed, CBJ, with its mechanism of backjumping, already integrates a clever tool to reduce redundancies. Therefore, even though CBJ-TD improves results, the gap is not so important. Finally, we notice that DR does not fare well when compared to basic algorithms such as CB and CBJ. It seems to us that this is due to the persevering heuristic [PV04] which drastically reduces possible choices when unassigning variables. Chances are there will be few variables causing the failure in the current cluster and among those it is unlikely that there will be two or more variables with the least occurrences in the explanations. This might even lead to some simple infinite loops between two or three variables.

We also ran experiments on randomly generated structured instances, with the method described in [JNT06]. We draw the same conclusions than with the CSP competition benchmark, but we also noted two interesting facts. First, DR fares as well as the best
algorithms (BTD, CBJ, CBJ-TD) on consistent instances with very dense clusters. We attribute this to the freedom given to DR, as it will be able to make interesting choices when backtracking, as there should be many variables to choose from, thus leading quickly to a solution. Second, CBJ performs worse on inconsistent instances with very dense clusters than on any other type of tested instances. This is most likely due to the fact that it will mimic the behaviour of CB, as every variable in the cluster is likely to cause a failure. Interestingly, CBJ-TD does not have this drawback.

5 Conclusion and Further work

We have described a generic framework for solving CSPs, which is an extension of the framework proposed by Pralet and Verfaillie in [PV04] in order to allow us to exploit the problem structure during the search and evaluate its interest when using different approaches for exploring the search space: chronological backtracking, Conflict directed BackJumping and Decision Repair. We have experimentally compared these different methods on structured binary instances, and we have found that exploiting the structure helps a lot for reducing redundancies when solving of an instance if this instance is "hard" enough. However, it can degrade the results for CSP instances "easily" solved by a clever traversal of the search space thanks to a good variable ordering heuristic.

The DR instantiation follows the persevere heuristic proposed in [PV04]. Experimental results have shown that this heuristic may lead to an over-intensification of the search process. It could be improved by adding some diversification mechanisms such as, for example, weighting the choice of the variable to unassign with the number of their occurrences in the explanations. This would prevent simple infinite loops.

There is still room for improvements in the way algorithms exploit the tree decomposition. In particular, we could let the variable heuristic decide which cluster to go first and give it more freedom during the search. It could go to the most difficult part first, thus recording less (no)goods in the solving process. It would also be more likely to find a solution or to prove inconsistency faster.

We plan to define an instantiation of the filter function which maintains Arc Consistency, and to extend our framework to non binary CSPs.

References