Efficient Recognition of Acyclic Clustered Constraint Satisfaction Problems[*]

Igor Razgon and Barry O'Sullivan

Cork Constraint Computation Centre
Department of Computer Science, University College Cork, Ireland
{i.razgon,b.osullivan}@cs.ucc.ie

Abstract. In this paper we present a novel approach to solving Constraint Satisfaction Problems whose constraint graphs are highly clustered and the graph of clusters is close to being acyclic. Such graphs are encountered in many real world application domains such as configuration, diagnosis, model-based reasoning and scheduling. We present a class of variable ordering heuristics that exploit the clustered structure of the constraint network to inform search. We show how these heuristics can be used in conjunction with nogood learning to develop efficient solvers that can exploit propagation based on either forward checking or maintaining arc-consistency algorithms. Experimental results show that maintaining arc-consistency alone is not competitive with our approach, even if nogood learning and a well known variable ordering are incorporated. It is only by using our cluster-based heuristics can large problems be solved efficiently. The poor performance of maintaining arc-consistency is somewhat surprising, but quite easy to explain.

1 Introduction

Solving real-world Constraint Satisfaction Problems (CSPs) can prove difficult for a number of applications, such as scheduling [17], frequency assignment problems [13], multi-commodity flow congestion control [10], and protein structure prediction [20]. Problems of this nature are typically posed as a network over which a set of constraints are defined. For example, for multi-commodity flow, the nodes in the network correspond to locations (warehouses and shipment locations), and the edges in the network correspond to links between locations (roads). The constraints specify capacities of links, among other things.

The complexity of these network-structured problems can be specified in terms of their graph parameters. In general, the overall problem is NP-hard; however, the complexity can be defined as being exponential in the tree-width of the network [4]. Roughly, the more tree-like the network is, the easier it is to solve. Tree-structured CSPs are important in that inference is polynomial. Inference in tree-structured CSPs has been heavily studied in the literature. Within the CSP community, it has been addressed in [7,3,5].

Because of the efficiency of inference in tree-structured CSPs, researchers have developed several methods for converting a CSP Z into a tree-structured CSP Z'. These

[*] This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886).

methods include using tree-decomposition algorithms [1], and then performing inference on Z' [11,6], or performing a structural compilation of the problem [18].

In this paper we propose a novel approach to efficiently solve clustered CSPs whose meta-constraint graph (the graph in which each cluster is replaced by a meta-variable) is *close* to a tree but not completely acyclic. This approach does not rely on a compilation of the problem. Instead we develop a class of search heuristics that can exploit the clustering in the problem to help dramatically improve the efficiency of search. Using these search heuristics we propose an efficient algorithm for solving clustered CSPs that combines constraint propagation and nogood recording. Our main result is that the proposed method *obliviously recognizes* (without spending any computational effort to do the recognition) when the meta-constraint graph of the problem becomes acyclic and, if that happens, solves the underlying CSP efficiently. Our algorithm assumes that the clustered structure of the CSP has either been identified by the user, or in a preprocessing step; usually this is a simple task to approximate manually. An algorithm that exploits our search heuristics, with nogood learning and constraint propagation, can be seen as exploiting a backdoor set of variables [19] that are not equivalent to a cycle cutset [5]. Our experiments show that MAC augmented with nogood learning and a good fail-first heuristic is significantly out-performed by a solver based on our cluster-based search heuristics when tested on clustered CSPs with a large number of variables.

The remainder of the paper is organized as follows. In Section 2 we introduce our notation. In Section 3 the basic algorithm is presented in terms of constraint propagation using forward checking, along with a deep theoretical analysis of it. We present a generalization of the algorithm in Section 4 that incorporates MAC, and also a simplified nogood recording scheme. An empirical evaluation of the algorithm is presented in Section 5. Finally, in Section 6 we make some concluding remarks and briefly outline our future work.

2 Background

2.1 Terminology

A Constraint Satisfaction Problem (CSP) Z is a triplet (V, D, C), where V is a set of *variables*, and D is the set of domains of *values* for each variable. We use the notation $\langle u, val\rangle$ to say that val is a value of variable v. A *tuple* of values is a set of values of different variables. If $\langle u, val\rangle$ belongs to a tuple T, we say that $\langle u, val\rangle$ is the *assignment* of u in T. The last item, C, of Z is the set of *constraints*. We represent each constraint $c \in C$ as a set of *forbidden* tuples of values to the variables constrained by c.[1] A *partial solution* is a tuple of values that contains *no* forbidden tuple as a subset. A *complete solution* or just a *solution* is a partial solution assigning all the variables of V. To *solve* a CSP is to find one of its solutions or to prove that no solution exists.

Let V' be a set of variables assigned by a forbidden tuple T. We say the variables of V' are *constrained*. If all the forbidden tuples have length 2, we get a *binary* CSP. The forbidden tuples of a CSP are often referred to as *conflicts*.

[1] We employ an unusual definition of a constraint because it is more convenient for our discussion.

A tuple T of values is a *nogood* if it is a forbidden tuple or if any extension of it to a tuple that assigns all the variables contains a forbidden tuple. In particular, if a partial solution is a nogood then it cannot be extended to a full solution.

2.2 The FC-EBJ Algorithm

In this paper we design CSP *solvers* that, besides solving a CSP, achieve some theoretical guarantee on their runtime. We assume the reader is familiar with constraint solvers like Forward Checking (FC) [9] and Maintaining Arc-Consistency (MAC) [16]. All the constraint solvers maintain additional data structures where they keep the results of intermediate computation. We introduce some additional terminology related to these data structures.

The *run* of a constraint solver consists of a number of iterations where it enumerates partial solutions in order to extend them to a full solution or to prove that none exists. If we consider a particular iteration occurring during the execution of a solver then the partial solution P that the solver tries to extend at that iteration is called the *current partial solution*. At the considered iteration, some values may be temporarily removed from their domains because the solver detected that they cannot be utilized to extend the current partial solution. The values that are *not* removed from their domains are called *valid* values at that iteration. The set of valid values of variable v is called the *current domain* of v.

As stated above, a value $\langle u, val\rangle$ is removed from the current domain of u because $P \cup \{\langle u, val\rangle\}$ cannot be extended to a full solution, in other words, it contains a nogood T. The set $T \setminus \{\langle u, val\rangle\}$ (which is a subset of the current partial solution) is called an *eliminating explanation* of $\langle u, val\rangle$.

The only reason why a complete constraint solver removes a value from its current domain is that it has found (implicitly or explicitly) an eliminating explanation for it. Some algorithms, like FC and MAC, do not record eliminating explanations explicitly but there are other algorithms, like DBT [8] or MAC-DBT [12], that do. A simple way to transform FC into an algorithm maintaining eliminating explanations is to introduce the following three modifications (in all the items below P denotes the current partial solution).

– If a value $\langle u, val\rangle$ is removed because $P \cup \{\langle u, val\rangle\}$ contains a forbidden tuple T, $\langle u, val\rangle$ is associated with the eliminating explanation $T \setminus \{\langle u, val\rangle\}$.
– If a value $\langle u, val\rangle$ is removed by backtracking caused by the empty current domain of some variable v, the eliminating explanation of $\langle u, val\rangle$ is the union of elimination explanations of all values of a variable v minus $\langle u, val\rangle$.
– A removed value is restored in the current domain only when its eliminating explanation becomes *obsolete*, i.e. it is no longer a subset of P.

It follows from the last property that this algorithm possesses the ability to backjump [14]. For example, if the domain of the currently unassigned variable v is wiped out, but the last assignment in the eliminating explanations of the values of v is $\langle u, val\rangle$ assigned 10-th to the last, the domain of v will remain empty until 10 consecutive backtracks (or one backjump of length 10) are performed. Thus the algorithm may be viewed as a modification of FC-CBJ [14], which maintains separate "conflict sets" for every value. We term this algorithm FC-EBJ, replacing Conflict-Directed Backjumping by Explanation-Directed Backjumping. FC-EBJ is almost analogous to the CCFC- algorithm [2] with two differences. First, the description of CCFC- is not based on eliminating explanations which makes it less convenient for our discussion. Second, CCFC- is formulated for binary constraints only, while FC-EBJ can solve any CSP.

3 Recognizing Clustered Acyclic CSPs

Consider a CSP whose variables are partitioned into clusters specified by the user. The given CSP is *clustered acyclic* if, after contraction of the clusters so that they become single vertices, the constraint graph of the given CSP is transformed into a tree. A clustered acyclic CSP can be solved with a complexity equivalent to that of solving the CSP associated with the largest cluster multiplied by the number of clusters, which is much faster than traditional backtracking if the clusters are small.

There are CSPs corresponding to real-world problems that can be divided into relatively small clusters such that the constraint graph resulting from the contraction of these clusters is close to a tree, some of which were discussed in Section 1. Therefore, it would be worthwhile to design a search algorithm that can *recognize* that the CSP induced by the *current* domains of unassigned variables is a clustered acyclic CSP and, if that happens, solves the given CSP efficiently. A straightforward approach to do that is to apply a checking procedure after every instantiation made by the search algorithm. However, this approach requires additional time for checking the desired property and thus might be not worthwhile. In this section we propose an alternative approach for recognizing clustered acyclic CSPs. In particular, we show that FC-EBJ combined with nogood learning and guided by a specially designed variable-ordering heuristic *obliviously recognizes* clustered acyclic CSPs without spending additional time or memory.

The starting point for the design of the proposed method was the observation that FC-EBJ solves a tree-like CSP polynomially if the variables are assigned in a depth-first search (DFS) manner with respect to the constraint graph of the given CSP. Next, we observed that FC-EBJ can benefit from exploring the constraint graph in a DFS-like manner even if the given CSP is not acyclic. In particular, let P be the current partial solution maintained by FC-EBJ. If the residual CSP (the result of projecting the underlying CSP to the unassigned variables and removing values incompatible with P) is acyclic then FC-EBJ takes polynomial time to check whether P can be extended to a full solution or it is a nogood. What is most interesting is that FC-EBJ neither “knows” that the residual CSP is acyclic nor applies any additional effort to recognize that. The algorithm just proceeds exploring the search space as before and the polynomial time complexity of exploring the residual CSP “comes” automatically. We say that FC-EBJ with a special ordering heuristic *obliviously recognizes* acyclic CSPs. Our last observation was that if this ordering heuristic is combined with nogood learning then the resulting version of FC-EBJ obliviously recognizes acyclic clustered CSPs.

It is worth noting that the proposed method differs from the traditional approach of compiling clusters and treating them as meta-variables with the set of solutions to the clusters as meta-values[2]. Instead, we implicitly apply a "lazy" compilation. That is, solutions to clusters may be learned as nogoods if the underlying CSP is hard and, in the worst case, the proposed algorithm has the same time and space complexity as the traditional method. However, if the underlying CSP is not too hard, it might be solved by the proposed algorithm without any compilation at all. This "compile when needed" paradigm results in huge time savings, as witnessed by the work of Bayardo and Miranker [3].

We begin the description of the proposed method from the formal definition of an acyclic clustered CSP.

Definition 1 (Clustered Acyclic CSP). *Let $Z = (V, D, C)$ be a CSP and let $SV = \{V_1, \dots V_l\}$ be a partition of V. Assume that each constraint in C constrains variables that belong to at most two different elements of SV. Let H be a constraint graph of Z i.e. a hypergraph with the vertices corresponding to V and the hyperedges corresponding to the subsets of V, which are scopes of the constraints of C. Consider the graph H' obtained from H by contracting the sets $V_1, \dots V_l$ to single vertices $v_1, \dots v_l$, removing the loops, and replacing the multiple edges by single edges. (According to our assumption all the edges of the resulting graph are binary). We denote by H' the clustered graph of Z with respect to SV. We say that Z is a clustered acyclic CSP with respect to SV, if H' is acyclic.*

Figure 1 illustrates the notion of acyclic clustered CSP. The dots represent CSP variables. All the constraints are binary. The constrained variables are connected by lines. The set of clusters $SV = \{\{V_1, V_2, V_3\}, \{V_4, V_5, V_6\}, \{V_7, V_8, V_9\}\}$. Clearly, as a result of contraction of these clusters, we get an acyclic constraint graph.

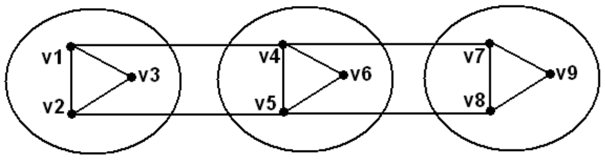


Fig. 1. Illustration of an acyclic clustered CSP

To proceed, we need to extend our notation regarding acyclic clustered CSPs. Let Z and SV be as in Definition 1. We denote by $M(Z, SV)$ the *maximal number of partial solutions* of a CSP created by the variables of V_i (i.e. the CSP consisting of the variables of V_i, their domains and the forbidden tuples assigning the variables of V_i only) among all $V_i \in SV$. Let v be a variable of Z. We denote by $Cl(v)$ the cluster of SV that v belongs to.

Now, we combine the FC-EBJ solver (Section 2.2) with a nogood learning procedure. Everytime a value $\langle u, val \rangle$ is removed, and the removal is justified by an eliminating explanation P', the resulting algorithm records the nogood $P' \cup \{\langle u, val \rangle\}$ in the store of nogoods. Nogoods are *discarded* from the store according to the following algorithm.

[2] In this setting recognizing clustered acyclic CSPs is equivalent to recognizing acyclic CSPs.

Let $(\langle v_1, val_1\rangle, \dots, \langle v_k, val_k\rangle)$ be a nogood recorded in the store. Assume the assignments are listed in the chronological order of their appearance in the current partial solution and the assignments that do not belong to the current partial solution are listed at the end ordered arbitrarily. Assume that for some $l < m < k$, $Cl(v_{l+1}) = \dots = Cl(v_m)$ and $Cl(v_{m+1}) = \dots = Cl(v_k)$ but $Cl(v_m) \neq Cl(v_k)$. Then the nogood is stored until the assignment $\langle v_l, val_l\rangle$ becomes obsolete (removed from the current partial solution). In other words, a nogood is stored until its obsolete part assigns variables of 3 or more clusters. All the time the nogood is stored, it is considered by the algorithm as a forbidden tuple. We call the resulting solver FC-EBJ-NL (NL is the abbreviation of Nogood Learning).

Proving the properties of FC-EBJ-NL, we extensively use the notions defined below (recall that P always denotes the current partial solution).

Definition 2 (Actual Forbidden Tuple). *Let S be a forbidden tuple (either original or dynamically acquired). We say that S is actual if it satisfies one of the following two conditions:*

- *There is a removed value $\langle u, val\rangle \in S$ such that $S \setminus \{\langle u, val\rangle\}$ is the eliminating explanation for $\langle u, val\rangle$.*
- *All the variables of $V(S \setminus P)$ (recall that P is the current partial solution) are unassigned and all the values of $S \setminus P$ belong to the current domains of their variables.*

Thus a forbidden tuple is *actual* if it is either used for pruning domain values or still may be violated by an extension of the current partial solution.

Definition 3 (Current CSP and Current Set of Clusters). *Let P be the current partial solution of FC-EBJ-NL applied to a CSP Z with a set of clusters SV.*

- *The current CSP Z' has the set of variables $V(Z) \setminus V(P)$. The domain of each variable $u \in V(Z')$ is the* current domain *of this variable. The constraints are defined as follows: for each actual forbidden tuple Q of Z including the values of Z', the tuple $Q \setminus P$ is forbidden in Z'. (The definition includes the forbidden tuples dynamically stored during the solution process at the considered moment of execution of FC-EBJ-NL.)*
- *The current set of clusters SV' is obtained from SV by removing the variables assigned by P from each cluster of SV.*

The proposed class of ordering heuristics includes any ordering heuristic that satisfies the following three conditions.

1. The first variable is selected arbitrarily.
2. Let u be the last assigned variable and let V_i be the cluster containing u. Assume that V_i contains an unassigned variable. Then the variable assigned next to u belongs to V_i.
3. Assume that each cluster is either fully assigned or fully unassigned. Let V' be an unassigned cluster *sharing* an actual forbidden tuple S (either original or dynamically acquired) with the *chronologically latest* possible assigned cluster V'' (in the

sense that S involves variables of both clusters). Then select any variable of V'. If there is no cluster V' as above, i.e. no assigned cluster shares an active forbidden tuple with an unassigned cluster, then select an arbitrary variable.

The last item of the above description has a much simpler formulation for the binary case: select a variable whose domain values are removed because of conflicts with the *latest possible* assigned cluster. We call the class of heuristics satisfying the above conditions LCC, which is the acronym for Last Conflicting Cluster.

The main theorem regarding FC-EBJ-NL using a LCC heuristic is as follows.

Theorem 1 (Main Result). *Assume that FC-EBJ-NL guided by a LCC heuristic is applied to a CSP Z with a collection of clusters SV. Let P be the current partial solution, let Z' be the current CSP, and let SV' be the current set of clusters. Assume that Z' is a clustered acyclic CSP with respect to SV'. Then the algorithm checks whether P can be extended to a full solution (and returns such a solution if yes, or refutes P if no) in time $M(Z', SV')^2 * (|SV'| - 1)$.*

Lemma 1. *Assume that FC-EBJ-NL is applied to a CSP Z with a set of clusters SV and consider an intermediate iteration that occurs during the processing of Z. Let P be the current partial solution, Z' be the current CSP, and SV' be the current set of clusters. Then the number of iterations spent by FC-EBJ-NL in order to check whether P can be extended to a full solution equals the number of iterations spent by FC-EBJ-NL in order to solve Z'.*

Proof of Lemma 1 is elementary but quite lengthy, we omit the proof due to space constraints. According to Lemma 1, Theorem 1 is implied by the following theorem.

Theorem 2. *Let Z be a clustered acyclic CSP with respect to a set of clusters SV. Then FC-EBJ-NL guided by a LCC heuristic solves Z performing at most $M(Z, SV)^2 * (|SV| - 1)$ backtracks.*

Proof of Theorem 2. To prove Theorem 2, we recall the way the clusters of Z are assigned by the LCC heuristic. After the full assignment of the first cluster V_1, the heuristic selects the next cluster V_2 such that V_2 "shares" nogoods with the assignments of V_1, the next cluster V_3 is selected on the same principle with respect to V_2 and, if impossible, with respect to V_1. Proceeding analogously, having assigned clusters $V_1, \ldots, V_k$, the algorithm selects the next cluster V_{k+1} so that it is constrained with the assignments of an already assigned cluster V_i, $1 \leq i \leq k$, such that i is as large as possible.

However, the above process is not always successful. It may happen that after assigning clusters $V_1, \ldots, V_k$ no unassigned cluster is constrained with the assignments of these clusters. In this case, the clusters $V_1 \ldots V_k$ constitute a *link* and the algorithm starts a new link. The partial solution P consists of a number of consecutive links. Each link has the *root*, i.e. the cluster, which is fully assigned first. Each cluster V_i of a link which is not the root of the link has a *parent*, i.e. the cluster which *certified* the addition of V_i to the link (in the last item of the description of the LCC heuristics, cluster V'' certifies the addition of cluster V' to a link). This structure of the current partial solution allows us to prove the following lemma.

Lemma 2. *Assume that FC-EBJ-NL guided by the LCC heuristic discards a value $\langle u, val\rangle$ at some state during its execution and associates it with the eliminating explanation T. Then $T' = T \cup \{\langle u, val\rangle\}$ assigns at most two clusters. Moreover, if a cluster V' other then $Cl(u)$ is assigned by T' then*

– *$Cl(u)$ is not assigned by the current partial solution or V' is the parent of $Cl(u)$;*
– *V_i and $Cl(u)$ are adjacent in the cluster graph of Z.*

Proof of Lemma 2 (sketch). We consider the chronological sequence of eliminating explanations generated by FC-EBJ-NL and apply induction on this sequence. Let T' be the first eliminating explanation generated by FC-EBJ-NL. Clearly, T' is a forbidden tuple of Z, hence the adjacency condition holds automatically for this case. According to the structure of Z', T' assigns at most two clusters. Assume that $Cl(u)$ is partially assigned and let us show that if T' assigns *exactly* two clusters then it assigns, besides $Cl(u)$, the parent V_i of $Cl(u)$. Assume that T' assigns a cluster V_j other than V_i. Observe first that V_i and V_j are both fully assigned.

If V_i and V_j belong to the same link then, according to the behaviour of the LCC heuristic, there is a path from V_i to V_j in the clustered tree which includes only assigned clusters of their link. Since T' is the first nogood generated by FC-EBJ-NL, the next cluster is selected as sharing an *original* forbidden tuple with the previous clusters. Therefore clusters that get to the same link induce a connected subgraph of the clustered graph of Z. On the other hand, there is a path from V_i to V_j through $Cl(u)$, which is not fully assigned (u is unassigned at the moment $\langle u, val\rangle$ is removed). It follows that there are two paths from V_i to V_j in the clustered tree, a contradiction.

If V_i and V_j belong to different links then T' is a forbidden tuple connecting clusters V_j and $Cl(u)$. If at the moment the link of V_j is fully assigned, all the values of T' are valid then we get contradiction to that fact that the new link is started. If some of the values of T' are removed, they are associated with eliminating explanations involving the variables of the link of V_j or an earlier link. In any case, it is a contradiction that $Cl(u)$ belongs to a link assigned *later* than the link of V_j.

Consider now a nogood T' with the assumption that the eliminating explanations generated prior to it satisfy the conditions of the lemma. The following claims hold at the moment of generating T' (their proofs are omitted due to space constraints).

1. Each cluster of an existing link is adjacent to its parent in the clustered graph of Z.
2. Let V_j and V_k be two clusters assigned by a stored actual nogood S and assigned by the current partial solution. Then one of them is the parent of the other.

Given the above observations, we prove the present lemma for each possible way a new nogood T' may be generated.

– **T' is an original or acquired forbidden tuple.** By the structure of Z' and the induction assumption, T' assigns at most two clusters. If T' assigns *exactly* two clusters, assume that $Cl(u)$ is assigned by the current partial solution and we prove that the other assigned cluster is the parent V_i of $Cl(u)$. If we assume that the other assigned cluster is some V_j that belongs to another link then we get a contradiction, analogously to the basic case. If we assume that the other assigned cluster belongs to the same link as $Cl(u)$ then applying the first observation from the above list and the induction assumption, we derive the existence of a cycle in the clustered graph of Z.

- **T' is the union of eliminating explanations of a variable v with the empty domain.** If $Cl(v)$ is assigned by the current partial solution then, by the second observation from the above list, the nogood associated with each value of v assigns the parent of V (if it assigns any cluster besides $Cl(v)$). Clearly, the union of the eliminating explanations has the same form. If $Cl(v)$ is not assigned by the current partial solution then assume that there are two nogoods T_1 and T_2 associated with values of v which assign distinct clusters, both different from $Cl(v)$. By the induction assumption, both these clusters are adjacent to $Cl(v)$ in the clustered graph of Z hence they belong to different links. Assume that T_1 involves an *earlier assigned* link. This means that there is an actual forbidden tuple involving variables of that link in contradiction to the fact that a new link has been started. □

Thus Lemma 2 proves that nogoods discovered by FC-EBJ-NL assign at most two clusters and if two clusters are assigned, they are adjacent in the clustered tree of the underlying CSP. For each pair of clusters there are at most $M(Z, SV)^2$ possible nogoods assigning those clusters. Also there are $|SV| - 1$ pairs of adjacent clusters. Hence, the number of nogoods of the above type is at most $M(Z, SV)^2 * (|SV| - 1)$. Each nogood, once recorded in the nogood store, is never removed from there because, to be removed, a nogood must assign at least three different clusters. It follows that no nogood is discovered by a backtrack more than once. Consequently, FC-EBJ-NL spends at most $M(Z, SV)^2 * (|SV| - 1)$ backtracks solving a clustered acyclic CSP. Thus we have proved Theorem 2 and, as a result, Theorem 1. ■

4 Enhancements of the Basic Recognition Algorithm

In this section we discuss two modifications of the FC-EBJ-NL algorithm presented in Section 3. The first modification introduces achieving arc-consistency as the constraint propagation method. The second modification simplifies the procedure of nogood learning.

4.1 Recognizing Acyclic CSP Together with MAC

Let us formulate the MAC-EBJ-NL algorithm. In addition to the propagation performed by FC-EBJ-NL, MAC-EBJ-NL removes *unsupported* values. More precisely, let u and v be two unassigned variables. If MAC-EBJ-NL detects that a value $\langle u, val\rangle$ of the current domain of u is inconsistent with all the values of the current domain of v (in the sense that $\langle u, val\rangle$ has binary conflicts with all these values either existing or produced by propagation of non-binary forbidden tuples), the following operations are performed: computing S, the union of eliminating explanations of all the values of v; removing $\langle u, val\rangle$ and associating it with eliminating explanation S; recording $S \cup \{\langle u, val\rangle\}$ in the nogood store.

We consider two versions of MAC-EBJ-NL. In the first version, which we call Clustered MAC-EBJ-NL, variables u and v, as above, must belong to the same cluster, i.e.

$Cl(u) = Cl(v)$. In other words, $\langle u, val\rangle$ is not removed if it is unsupported by variable v that does not lie in the same cluster as u does. The second version of MAC-EBJ-NL performs *full* MAC, checking any pair (u, v) of constrained variables.

It turns out that Clustered MAC-EBJ-NL preserves the property of FC-EBJ-NL, that is, recognizes a clustered acyclic CSP Z performing at most $M(Z, SV)^2 * (|SV| - 1)$ backtracks. Observe that the nogood T resulting from removing $\langle u, val\rangle$ unsupported by a variable v consists of $\langle u, val\rangle$ and the union of eliminating explanations of all the removed values of v. Applying the same induction principle as in Lemma 2, we may assume that the union of the eliminating explanations assigns, besides $Cl(v)$, at most one cluster V'. If $Cl(u) = Cl(v)$ then clearly the resulting nogood T preserves the same property.

Surprisingly enough, full MAC-EBJ-NL does not preserve this property of FC-EBJ-NL. The reason is that when a value $\langle u, val\rangle$ is removed as being unsupported by a variable v from *another* cluster, the nogood produced as a result does not necessarily satisfy the conditions stated in Lemma 2. In particular, if $Cl(v)$ is an assigned cluster and $Cl(u)$ is not the parent of $Cl(v)$ then, as a result of removing $\langle u, val\rangle$, the obtained nogood might contain three clusters: $Cl(v)$, the parent of $Cl(v)$, and $Cl(u)$. One can show that because of the above phenomenon, nogoods assigning an arbitrary number of clusters may be generated and a bad ordering of values can cause an exponential number of generated nogoods. We omit the complete proof of this fact due to space constraints.

4.2 Simplifying the Nogood Learning Procedure

FC-EBJ-NL uses quite a complicated mechanism for removing obsolete nogoods. This mechanism is useful for the proof of Theorem 1 but it might be difficult for the practical implementation. In this section we present an alternative nogood learning procedure and show that FC-EBJ combined with this procedure efficiently recognizes acyclic clustered CSPs.

The proposed mechanism of learning allocates K "slots" for nogoods given some predefined constant K. Initially all these slots are empty. The first learned nogood is stored in the first slot, the next nogood is stored in the second one, then in the third and so on. No nogood is discarded until all the slots are occupied. Once this happens, the first nogood stored after that goes to the first slot erasing the nogood stored there before that, the next one goes to the second slot with the same effect and so on. In other words, this is a mechanism of nogood learning that uses a store of a constant size and removes the "oldest" nogood in case of overflow.

The following proposition, whose proof is omitted due to space constraints, makes the above storage useful for our purposes.

Proposition 1. *Fix some two states during the execution of a constraint solver which uses the above nogood recording mechanism. Assume that at most K nogoods are stored by the solver between these two states. Then none of these K nogoods is erased until the second state is reached.*

Given a CSP Z with the set of clusters SV, set $K = M(Z, SV)^2 * (|SV| - 1)$. Assume that at some moment during the execution of FC-EBJ-NL, the current CSP is a clustered acyclic one. By Theorem 1, in order to recognize an acyclic CSP, FC-EBJ-NL detects at most K nogoods provided that no one of them is discarded during the process of recognition. According to the Proposition 1, the presented mechanism of nogood learning guarantees they are not discarded. It follows that acyclic clustered CSPs can be efficiently recognized given the presented nogood learning mechanism which is much simpler than the one described in Section 3.

5 Experimental Evaluation

In this section we report results of an empirical evaluation carried out in order to assess the practical merits of our approach to efficiently recognize acyclic clustered CSPs. We performed experiments on *clustered random problems*. We generated these problems using the following parameters: number of variables (num_var); domain size (dom_size); size of cluster ($size\_cluster$), which always divides num_var; the number of clusters ($num\_clusters = num\_var/size\_cluster$); additional connectivity ($add\_connect$), a number from 0 to 99; cluster density (cp_1); cluster tightness (cp_2); external density (ep_1); external tightness (ep_2).

Given these parameters, a CSP is generated by the following process:

1. **Create the graph of clusters GC.** Create $num\_cluster$ vertices $v_1, \dots, v_{num\_clusters}$. Then generate a tree on these vertices as follows. First the tree consists of vertex v_1. Assume that the current tree consists of vertices $v_1, \dots, v_i$. The new vertex v_{i+1} is connected to one of the existing vertices selected uniformly at random. Having created the spanning tree, additional edges are introduced as follows. For each pair of non-adjacent vertices of the tree, a number between 0 to 99 is chosen at random. If this number is smaller than $add\_connect$ then the corresponding edge is introduced.
2. **Fill each cluster with variables.** For each cluster, $size\_cluster$ variables are generated. One of these variables per cluster has $dom\_size/2$ values in its domain (or $(dom\_size-1)/2$ in case dom_size is odd). The domain size of the rest of the variables in a cluster is dom_size. The reason of introducing one variable per cluster with a smaller domain size will be clear when we present the experimental results.
3. **Create conflicts within clusters.** This is done analogously to the well-known generator of Prosser [15] given parameters cp_1 and cp_2 which are analogous to the parameters p_1 and p_2 introduced by Prosser, respectively. In particular, the parameter cp_1 serves as the probability that there is a constraint between two particular variables in the given cluster. If the constraint exists, the parameter cp_2 determines the probability of a conflict between two values of the given variables.
4. **Create conflicts between variables of different clusters.** For each pair of variables u and v that belong to different clusters *connected by an edge in GC*, parameter ep_1 serves as the probability that there is a constraint between u and v. If the constraint exists, the parameter ep_2 serves as the probability that the given pair of values, one from u, the other from v, are conflicting.

In our experiments we took $num\_var = 100$, $dom\_size = 10$, $size\_cluster = 10$. That is, the generated CSPs have 100 variables partitioned into 10 clusters, 10 variables in each one. The additional connectivity is selected to be 5, that is, the topology of the graph of clusters is close to a tree. We performed tests for two values of cluster density: 80% and 90%. We selected just these values because clusters must be dense "by definition". For cluster density of 80% we chose the cluster tightness of 30%, for cluster density of 90% the chosen cluster tightness is 25%. The external density is always 10%. The external tightness is the varied parameter.

The rest of the section is divided into 2 subsections. In the first subsection we compare MAC-based algorithms, in the second subsection FC-based algorithms are compared.

5.1 Comparison of MAC-Based Algorithms

We have tested the following MAC-based algorithms:

- MAC-EBJ, i.e. the solver that maintains arc-consistency, records eliminating explanations for the removed values but employs no nogood learning. The smallest-domain first (a Fail-First – FF) heuristic [9], i.e. the heuristic that selected the smallest domain first, was used to guide the search performed by MAC-EBJ.
- MAC-EBJ-NL with the FF heuristic (referred as MAC-EBJ-NL-FF), i.e. the solver like the previous one with the only difference that a nogood learning mechanism is employed.
- MAC-EBJ-NL with a LCC heuristic (referred as Full MAC-EBJ-NL-LCC). The ties are broken by the FF heuristic, i.e. every time when a set of variables may be selected by the LCC heuristic, the one with the smallest domain is selected from this set.
- The same solver as the previous one but performing Clustered MAC (referred as Clustered MAC-EBJ-NL-LCC).

The algorithms above are tested on the two sets of instances presented in the introductory part of the section. We now explain why the instances are designed so that there is one variable per cluster with a smaller domain. This is done in order to "fool" the FF heuristic. Without that trick, the FF heuristic guides the search pretty much like an LCC heuristic which make the comparison of LCC and FF heuristics senseless.

Figures 2 and 3 show the behaviour of the last three solvers on sets of instances presented at the introductory part of the section. All the solvers use the simplified nogood learning mechanism described in Section 4.2. The size of the store is 10000 nogoods. The computational effort is measured in the number of backtracks. For each set of parameters, the result is obtained as the average of 50 runs. A run is stopped if it takes more than 50000 backtracks.

According to our experiments, MAC-EBJ (the solver mentioned first) was unable to solve most of the problems in the allocated number of iterations, hence we do not illustrate its behaviour in the figures.

One can see that both Full and Clustered MAC-EBJ-NL-LCC essentially outperform MAC-EBJ-NL-FF. In particular, for density of 90% at the phase transition region, Full MAC-EBJ-NL-LCC performs about 8 times better and Clustered MAC-EBJ-NL-LCC performs about 3 times better then MAC-EBJ-NL-FF. Note also that Clustered MAC-EBJ-NL-LCC outperforms MAC-EBJ-NL-FF doing much less constraint propagation.

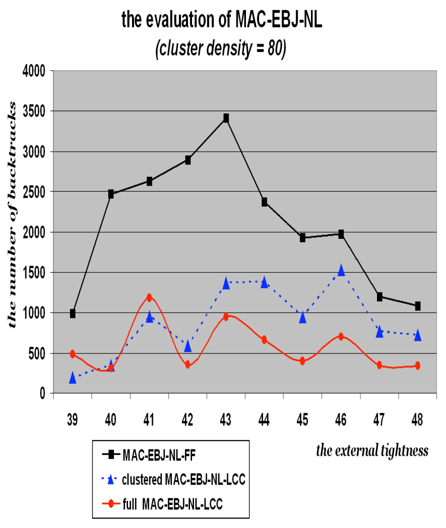


Fig. 2. Comparison of MAC-based algorithms, 80% density

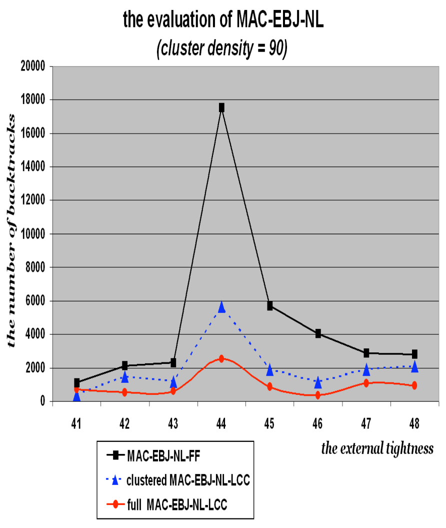


Fig. 3. Comparison of MAC-based algorithms, 90% density

Specifically, the Clustered MAC-EBJ-NL-LCC only propagates within its current cluster, but this is sufficient to improve upon MAC-EBJ-NL-FF. However, when we use our cluster-based search ordering heuristics we can fully propagate using MAC and gain some additional improvements, up to a factor of 3, over Clustered MAC-EBJ-NL-LCC.

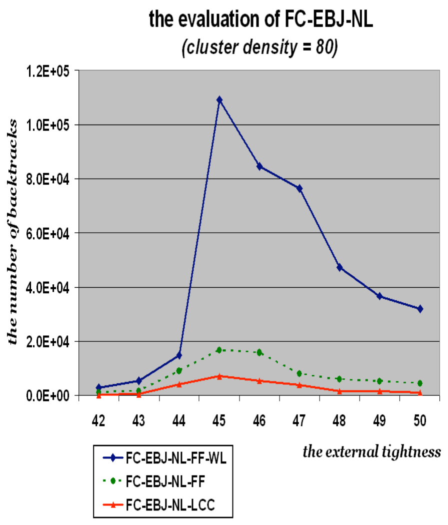
5.2 Comparison of FC-Based Algorithms

We have tested the following FC-based algorithms:

- FC-EBJ guided by FF heuristic that does not do any nogood learning. This solver is referred to as FC-EBJ-FF-WL (the last two letters abbreviate 'Without Learning').
- FC-EBJ-NL guided by the FF heuristic (referred as FC-EBJ-NL-FF).
- FC-EBJ-NL guided by the LCC heuristic (referred as FC-EBJ-NL-LCC). The breaking of ties is the same as in the case of MAC.

The algorithms that used the nogood learning mechanism are presented in Section 4.2. The algorithms have been tested on almost the same sets of instances as MAC-based algorithms with the only difference that for the cluster density 90% we have reduced the $add\_connect$ parameter to 4 because for value 5 of that parameter, all the algorithms took too long time to solve the resulting instances. The computational effort, as before, is measured in the number of backtracks, for each set of parameters 50 runs were applied with taking the average, the algorithms were stopped if they did more than 10^8 consistency checks. The experimental results are presented in Figure 4 and 5.

One can clearly observe that the nogood learning considerably improves the performance of the algorithms: both FC-EBJ-NL-FF and FC-EBJ-NL-LCC do much better than FC-EBJ-FF-WL. In particular, FC-EBJ-NL-LCC is about 12 times better than



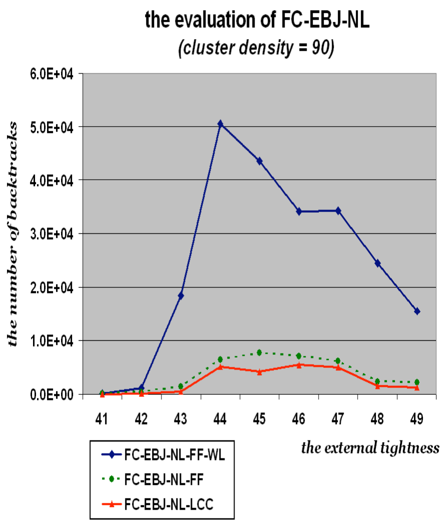


Fig. 4. Comparison of FC-based algorithms, 80% density

Fig. 5. Comparison of FC-based algorithms, 90% density

FC-EBJ-FF-WL for both considered groups of instances at the phase transition. These results correlate with the results of Bayardo and Miranker [3] on restricted nogood learning. Also FC-EBJ-NL-LCC is better than FC-EBJ-NL-FF but the rate of improvement is smaller than in the case when FC is replaced by MAC. It follows that the class of LCC heuristics are best applicable for MAC-based solvers.

6 Conclusion

It is well known that constraint graphs that are high clustered can be very challenging to solve using search-based methods. On the other hand, methods based on compilation require exponential space in the worst case. In this paper we have presented a novel algorithm for solving clustered CSPs efficiently. This algorithm can detect and exploit a tractable case automatically. Experimental results show the efficiency of our approach on large clustered CSPs.

References

1. Arnborg, S.: Efficient algorithms for combinatorial problems on graphs with bounded, decomposability–a survey. BIT 25(1), 2–23 (1985)
2. Bacchus, F.: Extending forward checking. Principles and Practice of Constraint Programming 35–51 (2000)
3. Bayardo, R.J., Miranker, D.P.: An optimal backtrack algorithm for tree-structured constraint satisfaction problems. Artif. Intell. 71(1), 159–181 (1994)
4. Bodlaender, H.L.: A tourist guide through treewidth. Acta Cybernetica 11, 1–21 (1993)

5. Dechter, R., Pearl, J.: Network-based heuristics for constraint-satisfaction problems. Artif. Intell. 34(1), 1–38 (1987)
6. Dechter, R., Pearl, J.: Tree clustering for constraint networks. Artif. Intell. 38(3), 353–366 (1989)
7. Freuder, E.C.: Complexity of k-tree structured constraint satisfaction problems. In: AAAI, pp. 4–9 (1990)
8. Ginsberg, M.L.: Dynamic backtracking. J. Artif. Intell. Res (JAIR) 1, 25–46 (1993)
9. Haralick, R.M., Elliott, G.L.: Increasing tree search efficiency for constraint satisfaction problems. Artif. Intell. 14(3), 263–313 (1980)
10. Harrelson, C., Hildrum, K., Rao, S.: A polynomial-time tree decomposition to minimize congestion. In: Proceedings of the ACM symposium on Parallel algorithms and architectures, pp. 34–43. ACM Press, New York (2003)
11. Jegou, P., Terrioux, C.: Hybrid backtracking bounded by tree-decomposition of constraint networks. Artif. Intell. 146(1), 43–75 (2003)
12. Jussien, N., Debruyne, R., Boizumault, P.: Maintaining arc-consistency within dynamic backtracking. In: Dechter, R. (ed.) CP 2000. LNCS, vol. 1894, pp. 249–261. Springer, Heidelberg (2000)
13. Koster, A.M.C.A., van Hoesel, S.P.M., Kolen, A.W.J.: Solving frequency assignment problems via tree-decomposition. Research Memoranda 036, METEOR (1999)
14. Prosser, P.: Hybrid Algorithms for the Constraint Satisfaction Problem. Computational Intelligence 9, 268–299 (1993)
15. Prosser, P.: Binary constraint satisfaction problems: Some are harder than others. In: ECAI, pp. 95–99 (1994)
16. Sabin, D., Freuder, E.C.: Contradicting conventional wisdom in constraint satisfaction. In: ECAI, pp. 125–129 (1994)
17. Sadeh, N.M., Sycara, K.P., Xiong, Y.: Backtracking techniques for the job shop scheduling constraint satisfaction problem. Artif. Intell. 76(1-2), 455–480 (1995)
18. Weigel, R., Faltings, B.: Compiling constraint satisfaction problems. Artificial Intelligence 115, 257–289 (1999)
19. Williams, R., Gomes, C.P., Selman, B.: Backdoors to typical case complexity. In: IJCAI, pp. 1173–1178 (2003)
20. Xu, J., Jiao, F., Berger, B.: A tree-decomposition approach to protein structure prediction. In: 2005 IEEE Computational Systems Bioinformatics Conference (CSB'05), pp. 247–256. IEEE Computer Society Press, Los Alamitos (2005)