Extracting MUCs from Constraint Networks

Fred Hemery and **Christophe Lecoutre** and **Lakhdar Sais** and **Frédéric Boussemart** [1]

Abstract. We address the problem of extracting Minimal Unsatisfiable Cores (MUCs) from constraint networks. This computationally hard problem has a practical interest in many application domains such as configuration, planning, diagnosis, etc. Indeed, identifying one or several disjoint MUCs can help circumscribe different sources of inconsistency in order to repair a system. In this paper, we propose an original approach that involves performing successive runs of a complete backtracking search, using constraint weighting, in order to surround an inconsistent part of a network, before identifying all *transition* constraints belonging to a MUC using a dichotomic process. We show the effectiveness of this approach, both theoretically and experimentally.

1 Introduction

A constraint network is said to be minimal unsatisfiable if and only if it is unsatisfiable and deleting an arbitrary constraint makes it satisfiable. Deciding whether a set of constraints is minimal unsatisfiable is well known to be DP-Complete [20]. It can be reduced to the SAT-UNSAT problem: given two CNF formulas ϕ and ψ, is ϕ satisfiable and ψ unsatisfiable? DP corresponds to the second level of the Boolean hierarchy. A problem in this class can be considered as the difference between two NP-problems.

On the practical side, when inconsistency is encountered, circumscribing the conflicting parts of a system can help the user understand, explain, diagnose and restore consistency. To illustrate the importance of the problem addressed in this paper, one can mention the well-known Radio Link Frequency Assignment Problem (RLFAP) which is often used as a benchmark in the CSP (Constraint Satisfaction Problem) community. This problem involves assigning frequencies to a set of radio links defined between pairs of transmitters in order to avoid interferences. To this end, one looks for a solution that minimizes the number of used frequencies. Circumscribing the unfeasible subnetwork areas (of minimal size) can help find new positions of the transmitters.

In the case of Boolean constraints (formula in Conjunctive Normal Form), finding minimal unsatisfiable sub-formula is an active research area. Tractable classes have been exhibited. Most of them are based on the deficiency of the formula (i.e. difference between the number of clauses and variables) [4, 7]. Also, recent advances in satisfiability checking has allowed successful extensions of SAT solvers for handling such a hard computational problem [3, 23, 15, 19].

In the context of constraint satisfaction, there is a significant amount of work dealing with the identification of conflict sets of constraints. Such sets, which can be built by recording explanations during search, are usually used to perform different forms of intelligent backtracking (e.g. [22, 8, 13]). However, there are only a few works really dedicated to the extraction of MUCs from constraint networks. An approach for the diagnosis of over-constrained networks has been proposed in [1] and a method to find all MUCs from a given set of constraints is presented in [9, 5]. This method corresponds to an exhaustive exploration of a so-called CS-tree, but is limited by the combinatorial explosion in the number of subsets of constraints. Finally, a divide and conquer approach has been proposed in [12] in order to extract from an over-constrained problem an explanation (or relaxation) using preferences given by the user.

In this paper, we propose an original approach to extract a MUC from a given constraint network. This approach consists of two stages. The first one exploits the conflict-directed variable ordering heuristic $dom/wdeg$ [2] in order to surround (and then extract) an unsatisfiable core by performing successive complete runs of a backtracking search algorithm. Search is restarted, while preserving constraint weighting from one run to the next one, until the size of the proved unsatisfiable core cannot be made smaller. Then, using a total order on the constraints based on their current weights, and following the principle introduced in [6], the second stage allows iteratively identifying the constraints of a MUC. Compared to *constructive* [6] and *destructive* [1] approaches which are respectively O($e.k_e$) and $\theta(e)$, the *dichotomic* approach that we propose is O($log(e).k_e$). Here, the complexity corresponds to the worst-case number of calls to the backtracking search algorithm, e denotes the number of constraints of the given constraint network and k_e denotes the number of constraints of the extracted MUC. We also relate this complexity with the one obtained by Junker [12].

The paper is organized as follows. First, we introduce some technical background. Then, we present the two stages of our approach: extracting an unsatisfiable core by exploiting constraint weighting and extracting a minimal core by identifying so-called *transition* constraints. Next, related work is discussed. Finally, before concluding, we present the results of an experimentation that we have conducted.

2 Technical Background

A Constraint Network (CN) P is a pair $(\mathscr{X}, \mathscr{C})$ where $\mathscr{X}$ is a finite set of n variables and $\mathscr{C}$ a finite set of e constraints. Each variable $X \in \mathscr{X}$ has an associated domain, denoted *dom*(X), which contains the set of values allowed for X. Each constraint $C \in \mathscr{C}$ involves a subset of variables of $\mathscr{X}$, called scope, and has an associated relation, denoted *rel*(C), which contains the set of tuples allowed for the variables of its scope. For any subset $S \subseteq \mathscr{C}$ of constraints of P, $P^{\uparrow S}$ will denote the constraint network obtained from P by removing all constraints of S and $P_{\downarrow S}$ will be equivalent to $P^{\uparrow(\mathscr{C}-S)}$.

A solution to a CN is an assignment of values to all the variables such that all the constraints are satisfied. A CN is said to be satisfiable iff it admits at least one solution. The Constraint Satisfaction Problem (CSP) is the NP-complete task of determining whether a given

[1] CRIL-CNRS FRE 2499, rue de l'université, SP 16, 62307 Lens cedex, France. email: {hemery,lecoutre,sais,boussemart}@cril.univ-artois.fr

CN is satisfiable. A CSP instance is then defined by a CN, and solving it involves either finding one (or more) solution or determining its unsatisfiability. To solve a CSP instance, a depth-first search algorithm with backtracking can be applied, where at each step of the search, a variable assignment is performed followed by a filtering process called constraint propagation. Usually, constraint propagation algorithms, which are based on some constraint network properties such as arc consistency, remove some values which can not occur in any solution. The algorithm that maintains arc consistency during search is called MAC. An unsatisfiable core corresponds to an unsatisfiable subnetwork of a CN.

Definition 1. *Let $P = (\mathscr{X}, \mathscr{C})$, $P' = (\mathscr{X}', \mathscr{C}')$ be two CNs. P' is an unsatisfiable core of P iff P' is unsatisfiable, $\mathscr{X}' \subseteq \mathscr{X} \wedge \mathscr{C}' \subseteq \mathscr{C}$.*

Different unsatisfiable cores of a given CN may exist. Those which do not contain any proper unsatisfiable core are said to be minimal.

Definition 2. *Let $P = (\mathscr{X}, \mathscr{C})$ be a CN and $P' = (\mathscr{X}', \mathscr{C}')$ an unsatisfiable core of P. P' is a Minimal Unsatisfiable Core (MUC) of P iff it does not exist any unsatisfiable core P'' of P' s.t. $P'' \neq P'$.*

To show the minimality of an unsatisfiable core, one can just check the satisfiability of any CN obtained when removing one constraint.

3 Extracting Unsatisfiable Cores

First, following the idea given in [1], we introduce an approach that allows removing some constraints (while preserving unsatisfiability). Then, we refine this approach by exploiting constraint weighting and (complete) restarts.

3.1 A Proof-based Approach

When the unsatisfiability of a CSP instance is proved by a filtering search algorithm, one can automatically extract a core that is guaranteed to be unsatisfiable. Indeed, it suffices to keep track of all the constraints that have been involved in the proof of unsatisfiability, that is to say, any constraint that has been used during search to remove, by propagation, at least one value in the domain of a variable. This principle was mentioned in [1] and can be related to the concept of implication graph used in SAT (e.g. see [17, 23]).

Let us examine how it works with MAC which maintains arc consistency by exploiting, for instance, an algorithm such as AC3 [16]. It involves successive revisions of arcs (pairs composed of a constraint and of a variable) in order to remove the values that are no more consistent with the current state. At the heart of the solver is then the function depicted in Algorithm 1. All values of the domain of the given variable that are not currently supported by the given constraint are removed (lines 2 to 4).

By introducing a data structure, denoted $active$, that allows associating a Boolean with each constraint, we are then in a position to extract an unsatisfiable core. The function $pcore$ depicted in Algorithm 2 allows such an extraction. Initially, all Booleans are set to $false$ (line 1). Then, the MAC solver is called (line 2), what involves successive revisions. Hence, whenever a revision is effective, the Boolean associated with the constraint is set to $true$ (line 5 of Algorithm 1). Finally, the function returns (line 3) the CN obtained from P by removing any constraint C such that $active[C]$ is $false$. It is important to remark that the network returned by $pcore$ is guaranteed to be unsatisfiable but not necessarily minimal.

Algorithm 1 revise(C : Constraint, X : Variable) : Boolean

```
1: $domainSize \leftarrow |dom(X)|$
2: for each $a \in dom(X)$ do
3:     if $seekSupport(C, X, a) = false$ then
4:         remove $a$ from $dom(X)$
5:         $active[C] \leftarrow true$
6: if $dom(X) = \emptyset$ then
7:     $wght[C] \leftarrow wght[C] + 1$ // used by some heuristics
8: return $domainSize \neq |dom(X)|$
```

Algorithm 2 pcore(P= $(\mathscr{V}, \mathscr{C})$: CN) : CN

```
1: $active[C] \leftarrow false, \forall C \in \mathscr{C}$
2: MAC($P$)
3: return $P^{\uparrow\{C \in \mathscr{C} | active[C] = false\}}$
```

3.2 A Conflict-based Approach

Even if the proof-based approach is an elegant approach, we have no idea about its practical efficiency. In other words, we cannot predict the size of the unsatisfiable core extracted by $pcore$. It is clear that the smallest the size is, the most efficient the approach is. Actually, as illustrated below, exploiting a conflict-directed variable ordering heuristic in order to push up an unsatisfiable core by performing successive runs makes the proof-based approach quite effective.

Heuristic dom/wdeg In [2], it is proposed to associate a counter, denoted $wght[C]$, with any constraint C of the problem. These counters are used as constraint weighting. Whenever a constraint is shown to be unsatisfied (during the constraint propagation process), its weight is incremented by 1 (see line 7 of Algorithm 1). The weighted degree of a variable X is then defined as the sum of the weights of the constraints involving X and at least another uninstantiated variable. The conflict-directed heuristic $dom/wdeg$ [2] involves selecting first the variable with the smallest ratio current domain size to current weighted degree. As search progresses, the weight of hard constraints become more and more important and this particularly helps the heuristic to select variables appearing in the hard part of the network. This heuristic has been shown to be quite efficient [2, 14, 10].

Illustrative Example Using $dom/wdeg$ allows efficiently proving the unsatisfiability of many instances. However, in order to obtain a proof of unsatisfiability of moderate size, one has to be aware that it is important to perform successive runs by restarting search several times. As an illustration, let us consider the problem of putting some queens and some knights on a chessboard as described in [2]. The instance with 6 queens and 3 knights involves 9 variables and 36 constraints and is unsatisfiable. In fact, we know that the subproblem corresponding to the 3 knights, and involving 3 variables and 3 constraints, is unsatisfiable. In a first phase, solving this instance with MAC-$dom/wdeg$ (i.e. MAC combined with the $dom/wdeg$ variable ordering heuristic) yields a proof of unsatisfiability integrating all constraints of the instance (that is to say, all Boolean $active$ have been set to $true$). However, solving again the same instance, using current weighting of the constraints as obtained after the first run, yields a new proof of unsatisfiability integrating only 9 constraints. An additional run furnishes the same result.

Figure 1 illustrates the evolution of such proofs of unsatisfiability. CNs are represented by constraint graphs where vertices correspond to variables and edges to binary constraints. Note that constraints ir-

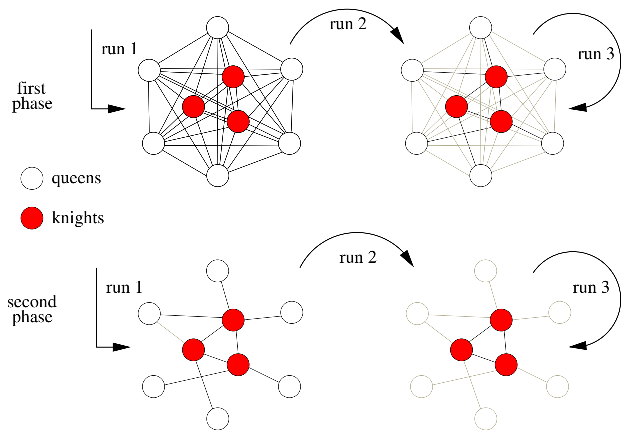

Figure 1. Evolution of the proof of unsatisfiability

relevant to unsatisfiability after one run are represented by dashed edges. Then, it is possible to refine the extraction after removing all the constraints which are not involved in the detected core, i.e., proof of unsatisfiability. Indeed, in a second phase, we obtain an unsatisfiable core that corresponds to the knights subproblem.

Exploiting Conflict-directed Heuristics As illustrated above, performing several runs of a MAC solver may be useful to surround an unsatisfiable core provided that a conflict-directed heuristic such as $dom/wdeg$ is used. This approach is depicted by Algorithm 3. Initially (line 1), the weight of all constraints is set to 1. Then, iteratively, MAC-$dom/wdeg$ is run (line 6) and the number of constraints found in the unsatisfiable core detected by the current run is counted (line 7). The iteration stops when the size of the current unsatisfiable core is greater than or equal to the size of the previous one. Remember that from one run to the next one, the $wght$ counters are preserved, which allows potentially concentrating the search to a smaller and smaller unsatisfiable core. Note that we can easily generalize this algorithm in order to perform several phases as mentioned in the illustration.

Algorithm 3 wcore(P= $(\mathscr{V},\mathscr{C})$: CN) : CN

```
1: $wght[C] \leftarrow 1, \forall C \in \mathscr{C}$
2: $cnt_{aft} \leftarrow +\infty$
3: **repeat**
4:     $active[C] \leftarrow false\ \forall C \in \mathscr{C}$
5:     $cnt_{bef} \leftarrow cnt_{aft}$
6:     MAC-$dom/wdeg(P)$
7:     $cnt_{aft} \leftarrow \mid \{C \in \mathscr{C} | active[C]\} \mid$
8: **until** $cnt_{aft} \geq cnt_{bef}$
9: return $P^{\uparrow\{C \in \mathscr{C} | active[C]=false\}}$
```

4 Extracting Minimal Unsatisfiable Cores

It is clear that the unsatisfiable core that can be extracted by using the function $wcore$ is not guaranteed to be minimal. In order to find a minimal core, it is necessary to iteratively identify the constraints that are involved in it. More precisely, we know that, given an unsatisfiable CN P and a total ordering of the constraints (to simplify, we shall consider the natural lexicographic order $C_1, C_2, \ldots, C_e$ of the constraints), there exists a constraint C_i such that $P_{\downarrow\{C_1,\ldots,C_{i-1}\}}$ is satisfiable and $P_{\downarrow\{C_1,\ldots,C_i\}}$ is unsatisfiable[2]. This constraint which clearly belongs to a minimal core of P will be called the *transition* constraint of P (according to the given ordering). Note also that any constraint C_j with $j > i$ can be safely removed.

4.1 Identifying the Transition Constraint

It is possible to identify the *transition* constraint of an unsatisfiable CN P by using a constructive approach, a destructive approach or a dichotomic one. Below, MAC(P) returns SAT (reps. $UNSAT$) iff P is satisfiable (resp. unsatisfiable), and the parameter k ($< |\mathscr{C}|$) indicates the number of transition constraints previously identified (the first k constraints of the current network). It will be meaningful later, but initially, just consider $k = 0$.

Constructive Approach The principle is to successively add the constraints of the given network until the current network becomes unsatisfiable. This approach is analog to the one introduced in [6] and is depicted by Algorithm 4.

Algorithm 4 csTransition(P= $(\mathscr{V},\mathscr{C})$: CN, k : int) : Constraint

```
1: **for** $i$ increasingly varying from $k+1$ to $|\mathscr{C}|$ **do**
2:     **if** MAC($P_{\downarrow\{C_1,\ldots,C_i\}}$) = UNSAT **then** return $C_i$
```

Destructive Approach The principle is to successively remove the constraints of the given network until the current network becomes satisfiable (note[2] that i can never reach 1). This approach has been introduced in [1] and is depicted by Algorithm 5.

Algorithm 5 dsTransition(P= $(\mathscr{V},\mathscr{C})$: CN, k : int) : Constraint

```
1: **for** $i$ decreasingly varying from $|\mathscr{C}|$ to $k+1$ **do**
2:     **if** MAC($P_{\downarrow\{C_1,\ldots,C_{i-1}\}}$) = SAT **then** return $C_i$
```

Dichotomic Approach Finally, it is possible to use a dichotomic search in order to find the transition constraint. This approach is depicted by Algorithm 6. At each step of the search, we know that the transition constraint of P belongs to $\{C_{min}, \ldots, C_{max}\}$.

Algorithm 6 dcTransition(P= $(\mathscr{V},\mathscr{C})$: CN, k : int) : Constraint

```
1: $min \leftarrow k+1$ ; $max \leftarrow |\mathscr{C}|$
2: **while** $min \neq max$ **do**
3:     $center \leftarrow (min + max)/2$
4:     **if** MAC($P_{\downarrow\{C_1,\ldots,C_{center}\}}$) = SAT **then** $min \leftarrow center + 1$
5:     **else** $max \leftarrow center$
6: return $C_{min}$
```

4.2 Extracting the Minimal Core

Using one of the functions described above allows finding the transition constraint C_i of an unsatisfiable network P, that is to say one element that belongs to a minimal core of P. To get a second element, we apply the same function on a new CN P' which is obtained from P by removing all constraints C_j such that $j > i$ (since unsatisfiability is preserved) and considering a new order of the constraints such that C_i is considered as the smallest element (a natural order can be preserved by simply renaming constraints). This process can be repeated until all constraints of the current network correspond to transition constraints that have been successively found. The principle of this iterative process has been described in [6, 11, 21]. It is

[2] We shall assume that no constraint C exists in P such that rel(C) = $\emptyset$.

depicted by Algorithm 7 which returns a MUC from the given network (just consider one of the tree approaches by replacing xx with cs, ds or dc). Note that we have to determine if the last constraint belongs to the MUC (lines 7 and 8).

Algorithm 7 xxMUC(P= $(\mathscr{V},\mathscr{C})$: CN) : CN

```
1: $P' \leftarrow P$ ; $k \leftarrow 0$
2: **while** $k < |\mathscr{C}'| - 1$ **do**
3:     $C_i \leftarrow xxTransition(P', k)$
4:     $k \leftarrow k + 1$
5:     $P' \leftarrow P'^{\uparrow\{C_j | j > i\}}$
6:     in $P'$, $tmp \leftarrow C_i$, $C_{j+1} \leftarrow C_j$ for $1 \leq j < i$, $C_1 \leftarrow tmp$
7: **if** MAC($P'^{\uparrow\{C_{|\mathscr{C}'|}\}}$) = UNSAT **then** return $P'^{\uparrow\{C_{|\mathscr{C}'|}\}}$
8: **else** return $P'$
```

The following proposition (whose proof is omitted) suggests that the dichotomic approach should be more efficient than the two other ones.

Proposition 1. *Let $P = (\mathscr{X},\mathscr{C})$ be an unsatisfiable CN. The worst-case number of calls to MAC is O($e.k_e$) for csMUC(P), $\theta(e)$ for dsMUC(P) and O($log(e).k_e$) for dcMUC(P). Here, $e = |\mathscr{C}|$ and k_e is the number of constraints of the extracted MUC.*

To be more precise, we obtain a worst-case number of calls to MAC by dcMUC bounded by $log_2(e).(k_e + 1)$ ($+1$ since we have to prove that the k_e transition constraints form a MUC). It can be compared with $QuickXplain$ whose worst-case complexity is $2k_e.log_2(e/k_e) + 2k_e$ [12]. In the worst-case, $dcMUC$ is better than $QuickXplain$ when $(log_2(k_e) - 1).2k_e/(k_e - 1) < log_2(e)$, that is to say, when the size of the extracted core is rather small. This condition holds for all the instances that have been tested in our experimentation. Also, considering the illustration given in [12] with $e = 2^{20}$ and $k_e = 2^3$, we obtain 288 calls with $QuickXplain$ against 180 calls with $dcMUC$.

Finally, for efficiency reasons, it is really important to use, in practice, a conflict-based approach (function $wcore$) before extracting a MUC (function $xxMUC$). This will be shown in Section 6. The methods that we consider are then (given a constraint network P):

- CS which corresponds to call csMUC(wcore(P))
- DS which corresponds to call dsMUC(wcore(P))
- DC which corresponds to call dcMUC(wcore(P))

Even if it does not explicitly appear above, we will consider that all constraints are renamed, before calling xxMUC, in such a way that the lexicographic order corresponds to the decreasing order of the current weights of the constraints. It allows to improve the methods by limiting the number of runs performed. Note also that we can arbitrarily bound the number of calls to MAC by $wcore$ in order to have Proposition 1 hold for CS, DS and DC. In practice, we have observed that the number of calls to MAC by $wcore$ is always low.

5 Related Work

On the one hand, research in extraction of unsatisfiable cores of constraint networks is rather limited. Bakker et al. [1] have proposed a method to extract a MUC (in the context of Model-Based Diagnosis). This method essentially corresponds to call dsMUC(pcore(P)). Our approach can then be seen as a refinement[3] of theirs since we propose a dichotomic approach (dcMUC) and a conflict-based preliminary stage (wcore) whose practical importance will be shown in Section 6. Some other works [11, 21] concern the identification of minimal Π conflict-sets where Π denotes a propagation operator. Roughly speaking, while extracting a MUC is an activity which is global to the network, extracting a Π conflict-set is an activity limited to a branch of the search tree. As a consequence, in order to keep some incrementality of the propagation process, the proposed algorithms in [11, 21] involves (at least, partially) a constructive schema. The (new) method QuickXplain [12] exploits a divide and conquer approach (which exploits a dichotomic process) and whose complexity has been discussed in Section 4.2. A similar approach, called XC1, has been proposed in [18] in a more general context.

On the other hand, research in unsatisfiable cores for propositional satisfiability (SAT) is quite active. Bruni and Sassano [3] have proposed an "adaptive core search" to recover a small unsatisfiable subformula. Clause hardness are evaluated thanks to an history search analysis. By selecting a fixed percentage of hard clauses, the current unsatisfiable core is expended or contracted until the core becomes unsatisfiable. In [23], using a resolution proof based approach, the Zchaff solver is extended for approximating an unsatisfiable core (i.e. the returned unsatisfiable core is not guaranteed to be minimal). Finally, Lynce and Marques-Silva [15] have proposed a model that computes a *minimum* unsatisfiable core (i.e. the smallest unsatisfiable core in the number of clauses).

6 Experiments

In order to show the practical interest of the approach described in this paper, we have conducted an experimentation, using for each run MAC-$dom/wdeg$, on a PC Pentium IV 2,4GHz 512Mo under Linux. Performances have been measured in terms of the number of runs (#runs) and the cpu time in seconds (cpu). We also indicate the number of constraints (#C) (and, sometimes, of variables (#V))) of instances and extracted cores.

Our experimentation has been performed wrt some random and real-world instances. The random instances correspond to the unsatisfiable instances of two sets, denoted ehi-85-297 and ehi-90-315, containing 100 easy random instances embedding a small unsatisfiable part. The real-world instances correspond to the two archives RLFAP and FAPP. The selected unsatisfiable instances of the Radio Link Frequency Assignment Problem (RLFAP) came from the CELAR (Centre electronique de l'armement) while the instances of the Frequency Assignment with Polarization Problem (FAPP) came from the ROADEF'2001 challenge. Most of these instances were used as benchmarks[4] for the first CSP solver competition.

Instance	pcore		wcore	
	cpu	#C	cpu	#C
qk-25-25-5-mul (#C = 435)	100.1	427	107.7	32
ehi-85-297-0 (#C = $4,094$)	2.93	$3,734$	3.01	226
$graph$-14-$f28$ (#C = $4,638$)	4.23	$3,412$	4.69	503

Table 1. Cost (cpu) and Size (#C) of cores extracted by $pcore$ and $wcore$

First, we have studied the practical interest of calling $wcore$ instead of $pcore$. Table 1 indicates the size of the cores extracted by both methods wrt some instances. One can observe that it is really worth using $wcore$ since the size of the extracted core can be very small and the additional cost of performing successive runs is not penalizing (the cost for $wcore$ is given for all performed runs). It is

[3] It is also important to note that weights introduced in [1] correspond to static preferences given by the user.

[4] `http://cpai.ucc.ie/05/Benchmarks.html`

illustrated with one queens-knights instance, one random 3-SAT instance and one RLFAP instance. Note that, for some other instances, the gap between the two methods is negligible.

Then, we have compared DC with CS and DS. On random EHI instances (mean costs are given in the first part of Table 2), the difference between DS and DC is not very important. It can be explained by the fact that the extracted core returned by $wdeg$ is already small (it has been observed on all EHI instances but not on all tested real-world instances). However, on RLFAP and FAPP instances (see second and third parts of Table 2), DC clearly outperforms CS and DS. Both, in terms of cpu and number of runs, DC is about 10 times more efficient than CS and DS. We have obtained this kind of results on about half the RLFAP and FAPP instances that we have tested.

Instance	Method	cpu	#runs	MUC	
				#V	#C
EHI instances (100 instances per series)					
$ehi-85-297$	CS	263	1284	19	32
$\#V=297$	DS	62	157	23	37
$\#C \approx 4100$	DC	45	153	19	32
$ehi-90-315$	CS	266	1294	19	32
$\#V=315$	DS	64	156	23	36
$\#C \approx 4370$	DC	46	154	19	32
RLFAP instances					
$graph13$-$w1$	CS	303	704	4	6
$\#V=916$	DS	257	338	4	6
$\#C=1479$	DC	39	55	4	6
$scen02$-$f25$	CS	145	588	10	15
$\#V=200$	DS	130	311	12	28
$\#C=1235$	DC	21	67	10	15
$scen09$-$w1$-$f3$	CS	468	576	6	8
$\#V=680$	DS	533	480	6	8
$\#C=1138$	DC	38	48	6	8
FAPP instances					
$fapp03$-300-5	CS	469	484	3	3
$\#V=300$	DS	333	290	3	3
$\#C=2326$	DC	57	37	3	3
$fapp06$-500-1	CS	394	393	3	3
$\#V=500$	DS	306	195	3	3
$\#C=3478$	DC	48	35	3	3
$fapp10$-900-1	CS	1206	688	4	4
$\#V=900$	DS	743	365	4	4
$\#C=6071$	DC	119	46	4	4

Table 2. Extracting a MUC from EHI, RLFAP and FAPP instances

Finally, we have used the three methods in order to iteratively remove, until satisfiability is reached, disjoint MUCs (i.e. MUCs that do not share any constraint) from two RLFAP instances. It takes about 5 minutes to reach this goal with DC. Remark that there are about 40 and 130 constraints involved in the set of disjoint MUCs extracted from $graph05$ and $scen11$-$f10$, respectively. It represents about 3.5% of the set of constraints of each instance.

Instance	Method	cpu	#runs	#MUCs
$graph05$	CS	$5,175$	$8,825$	5
$\#V=200$	DS	$1,996$	$2,010$	5
$\#C=1134$	DC	330	526	5
$scen11$-$f10$	CS	$1,491$	$3,840$	5
$\#V=680$	DS	$3,040$	$2,452$	5
$\#C=4103$	DC	263	562	5

Table 3. Extracting successive MUCS from RLFAP instances

7 Conclusion

In this paper, we have presented a new approach, denoted DC, that allows extracting MUCs from constraint networks. The originality of this approach is that it exploits the recent heuristic $dom/wdeg$ in order to surround an unsatisfiable core by performing successive complete runs of MAC, and a dichotomic search in order to identify the successive transition constraints belonging to a MUC. We have introduced both theoretical and practical arguments to support our approach. In particular, the worst-case number of calls to MAC of DC is bounded by $log_2(e).(k_e+1)$, and DC has appeared to be quite efficient with respect to real-world instances taken from the RLFAP and FAPP archives. Indeed, a MUC has been extracted from most of these instances in less than 1 minute. In comparison, it is worth mentioning that most of the instances that have been experimented in this paper can not be solved, in a reasonable amount of time, when using standard variable ordering heuristics (see [2, 14]).

Acknowledgments

This paper has been supported by the CNRS, the "programme COCOA de la Région Nord/Pas-de-Calais" and by the "IUT de Lens".

REFERENCES

[1] R.R. Baker, F. Dikker, F.Tempelman, and P.M. Wognum, 'Diagnosing and solving over-determined constraint satisfaction problems', in *Proceedings of IJCAI'93*, pp. 276–281, (1993).
[2] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, 'Boosting systematic search by weighting constraints', in *Proceedings of ECAI'04*, pp. 146–150, (2004).
[3] R. Bruni and A. Sassano, 'Detecting minimaly unsatisfiable subformulae in unsatisfiable SAT instances by means of adaptative core search', in *Proceedings of SAT'00*, (2000).
[4] H. Kleine Buning, 'On subclasses of minimal unsatisfiable formulas', *Discrete Applied Mathematics*, **107**(1-3), 83–98, (2000).
[5] M. Garcia de la Banda, P.J. Stuckey, and J. Wazny, 'Finding all minimal unsatisfiable subsets', in *Proceedings of PPDP'03*, (2003).
[6] J.L. de Siqueira and J.F. Puget, 'Explanation-based generalisation of failures', in *Proceedings of ECAI'88*, pp. 339–344, (1988).
[7] H. Fleischner, O. Kullmann, and S. Szeider, 'Polynomial-time recognition of minimal unsatisfiable formulas with fixed clause-variable difference', *Theoretical Computer Science*, **289**, 503–516, (2002).
[8] M. Ginsberg, 'Dynamic backtracking', *Artificial Intelligence*, **1**, 25–46, (1993).
[9] B. Han and S-J. Lee, 'Deriving minimal conflict sets by cs-trees with mark set in diagnosis from first principles', *IEEE Tansactions on Systems, Man and Cybernetics*, **29**(2), 281–286, (1999).
[10] T. Hulubei and B. O'Sullivan, 'Search heuristics and heavy-tailed behaviour', in *Proceedings of CP'05*, pp. 328–342, (2005).
[11] U. Junker, 'QuickXplain: conflict detection for abitrary constraint propagation algorithms', in *Proceedings of IJCAI'01 Workshop on modelling and solving problems with constraints*, pp. 75–82, (2001).
[12] U. Junker, 'QuickXplain: preferred explanations and relaxations for over-constrained problems', in *Proc. of AAAI'04*, pp. 167–172, (2004).
[13] N. Jussien and V. Barichard, 'The palm system: explanation-based constraint programming', in *Proc. of TRICS'00*, pp. 118–133, (2000).
[14] C. Lecoutre, F. Boussemart, and F. Hemery, 'Backjump-based techniques vs conflict-directed heuristics', *ICTAI'04*, pp. 549–557, (2004).
[15] I. Lynce and J.P. Marques-Silva, 'On computing minimum unsatisfiable cores', in *Proceedings of SAT'04*, (2004).
[16] A.K. Mackworth, 'Consistency in networks of relations', *Artificial Intelligence*, **8**(1), 99–118, (1977).
[17] J.P. Marques-Silva and K.A. Sakallah, 'Conflict analysis in search algorithms for propositional satisfiability', Technical Report RT/4/96, INESC, Lisboa, Portugal, (1996).
[18] J. Mauss and M. Tatar, 'Computing minimal conflicts for rich constraint languages', in *Proceedings of ECAI'02*, pp. 151–155, (2002).
[19] Y. Oh, M.N. Mneimneh, Z.S. Andraus, K.A. Sakallah, and I.L. Markov, 'AMUSE: A minimally-unsatisfiable subformula extractor', in *Proceedings of DAC'04*, pp. 518–523, (2004).
[20] C.H. Papadimitriou and D. Wolfe, 'The complexity of facets resolved', *Journal of Computer and System Sciences*, **37**, 2–13, (1988).
[21] T. Petit, C. Bessière, and J.C. Régin, 'A general conflict-set based framework for partial constraint satisfaction', in *Proceedings of SOFT'03 workshop held with CP'03*, (2003).
[22] P. Prosser, 'Hybrid algorithms for the constraint satisfaction problems', *Computational Intelligence*, **9**(3), 268–299, (1993).
[23] L. Zhang and S. Malik, 'Extracting small unsatisfiable cores from unsatisfiable boolean formulas', in *Proceedings of SAT'03*, (2003).