

Implementing Constraint Relaxation over Finite Domains using Assumption-based Truth Maintenance Systems

Narendra Jussien and Patrice Boizumault

École des Mines de Nantes. Département Informatique.
4 Rue Alfred Kastler. La Chantrerie.
F-44070 Nantes Cedex 03, France.
{Narendra.Jussien,Patrice.Boizumault}@emn.fr

Abstract. Many real-life Constraint Satisfaction Problems are over-constrained. In order to provide some kind of solution for such problems, this paper proposes a constraint relaxation mechanism fully integrated with the constraint solver. Such a constraint relaxation system must be able to perform two fundamental tasks: identification of constraints to relax and efficient constraint suppression. Assumption-based Truth Maintenance Systems propose a uniform framework to tackle those requirements. The main idea of our proposal is to use the ATMS to record and efficiently use all the information provided by the constraint solver while checking consistency. We detail the use of ATMS in our particular scheme and enlight their efficiency by comparing them with existing algorithms or systems (Menezes' IHCS and Bessière's DnAC4).

1 Introduction

Many real-life problems are over-constrained (time-tabling, CAD, ...). A constraint relaxation mechanism integrated with the constraint solver becomes necessary. Such a constraint relaxation system must be able to perform two fundamental tasks: identification of constraints to relax and efficient constraint suppression. This research area has been initiated by the Constraint Logic Programming community, from the works of Borning *et al.* [3] in the Logic Programming community and from Freuder's first theoretical framework [11] in the CSP community.

Several systems or algorithms have been proposed to provide a constraint relaxation system over finite domains such as IHCS [14], DnAC4 [1], DnAC6 [7], ... These propositions present two similarities : first, they all deal with a single point of constraint relaxation over finite domains – intelligent identification of constraint(s) to relax in IHCS or efficient constraint suppression in DnAC* – and second, they all record supplementary information provided by the constraint solver. It is worth noticing that this information is the same in all the cited systems – for each removed value in a variable domain, one records the constraint which first removed this value when checking consistency –, just the use of this information differs.

The aim of this paper is to propose a Constraint Relaxation scheme for CLP(\mathcal{FD}) languages integrating at the same time appropriate relaxation decision, identification of responsibilities and constraint suppression. Assumption-based Truth Maintenance Systems [5] propose a uniform framework to tackle those three requirements. The main idea of our proposal is to use the ATMS to record and efficiently use all the information provided by the constraint solver while checking consistency. The ATMS framework provides a direct management of constraint deletion, and a more precise identification of constraints to relax.

First, we briefly recall some definitions and results about Constraint Relaxation. Then, we present our framework for Constraint Relaxation over Finite Domains and recall basic results about ATMS. We describe the use of ATMS to implement our system and illustrate it using an example. Finally we give results from a first implementation and draw further works.

2 Constraint Relaxation

In order to deal with over constrained systems, various approaches have been developed for a few years. Constraint hierarchies and their use were introduced by Borning *et al.* [3], and Freuder [11] proposed a first theoretical framework. In this section, we briefly recall the basic concepts and review different proposals.

2.1 Basic Concepts

HCLP [3] introduces a **hierarchy** upon constraints. A **weight** is assigned to each constraint. This weight represents the relative importance of the constraint. This weight allows the setting of a partial order relation between constraints. Thus, constraints with no weight (which can be assimilated as a zero weight) are called **required** or **mandatory** constraints and those with a positive weight are called **preferred** constraints. Let us recall that the greater the weight, the less **important** the constraint is.

A substitution (values for variables) which satisfies all the required constraints and which satisfies the preferred constraints in the best possible way with respect to a given comparator is called a **solution**. Thus, a *maximal* (to a given criterion¹) sub-problem of the initial problem is searched. This sub-problem must have existing solutions.

2.2 Different approaches

A constraint relaxation problem can be handled in different ways.

- The problem can be seen as the modification of an already solved instance using repairing algorithms [12, 16]. This approach needs an existing solution in order to start and needs explicit operations, i.e. we have to tell the system which operation has to be executed (suppressing a constraint, adding a constraint, ...)

¹ This criterion is called the comparator in Borning's approach.

- A constraint relaxation problem can be solved using branch and bound methods with preferred constraints in the objective function. Those approaches use known Operations Research results and techniques [8]. In those approaches, the programmer or user cannot specify his labeling strategy and does not control the resolution which is guided by the objective function.
- Fages *et al.* [9] proposed a reactive scheme that can efficiently add and suppress constraints. An abstracted framework is presented in terms of transformations of CSLD trees. In this approach, the constraint(s) to suppress are selected by the user but not automatically detected by the system.
- The Constraint Relaxation problem can also be studied as the handling of a tower of constraints [3, 13]. For example, the HCLP(\mathcal{R}) system considers that the whole problem does not have any solution. Therefore it tries to find a solution respecting only the required constraints, and then refines the solution adding the preferred constraints level² by level until failure. Unfortunately, this approach is no more usable in the Finite Domains paradigm, because consistency techniques are local (eg., all the variables must be instantiated to insure the existence of a solution under the required constraints).

All those systems are not satisfactory for our purpose: providing an automated constraint relaxation system embedded in a Constraint Logic Programming language over Finite Domains.

3 Constraint Relaxation over Finite Domains

First, we review some works about efficient addition or suppression of constraints using dynamic CSP, and then about identifying constraint(s) to relax using intelligent backtracking. We show that even though not sharing the same objectives, they all keep the same *justification* system. Finally, we present our general framework.

3.1 Dynamic Arc Consistency

For a few years, the CSP community has been interested in Dynamic CSP. A few algorithms have been developed by Bessi ere (DnAC4 [1]) and Debruyne (DnAC6 [7]) to achieve dynamic arc-consistency which allows efficient addition and suppression of constraints.

DnAC4 extends AC4 by recording justifications during restrictions. The justification consists in recording for each removed value in the domain of a variable, the constraint which first removed it.

When deleting a constraint, DnAC4 finds which values must be put back in the domains thanks to the system of justifications. This procedure is more efficient than re-running an AC4 algorithm on the new problem from scratch.

² A set of constraints with a same weight forms a level in the hierarchy of constraints.

3.2 Identifying constraint(s) to relax

The Logic Programming community has been interested for a few years in Intelligent Backtracking. The results of this study have been embedded in the CLP(\mathcal{FD}) framework to identify responsibility of constraint for inconsistency and thus to identify constraints to relax.

When achieving arc-consistency, Cousin's approach [4] records the same justification as DnAC4. When the domain of a variable becomes empty, it is then easy to identify the appropriate constraints, but this notion of responsibility is very limited (we just identify an immediate responsible constraint). It is worth noticing that even though the recorded information is the same as DnAC4, Cousin does not make the same usage of this information. Cousin's approach appears as a rudimentary strategy for determining constraints to relax because of its short range view of the problem.

IHCS gives a deeper analysis of the situation. IHCS uses the same justification as DnAC4. Moreover, it defines a dependency relation between constraints: *a constraint C_a depends on a constraint C_b if the constraint C_b modifies the domain of a variable appearing in C_a* . The resulting graph can be used in order to identify responsibility of failures. Thus, when a constraint fails, it is easy to identify a set of constraints who could be responsible for the failure. This set is the transitive closure from the failing constraint in the dependency graph. The problem of the IHCS system is encountered in the fact that all the analysis is done from the *constraints* i.e. a lot of information is lost about *values* in domains of variables, then, the dependency analysis becomes too general and unnecessary possibilities of relaxation are handled.

It is worth noticing that all these orthogonal works lie upon the same notion of justification (the constraint who removed a value from the domain of a variable).

3.3 Our general framework

We state that Constraint Relaxation over Finite Domains relies on three main points which cannot be treated separately:

- *When to relax ?* When exactly during computation do we have to relax a constraint.

We have to start a relaxation process when a contradiction is raised ensuring the non existence of solution. Exploiting this approach is not so easy in a standard CLP(\mathcal{FD}) scheme. In such a scheme, there are two kinds of contradictions: a contradiction caused by a wrong value given during the labeling phase, and a contradiction due to the intractability of the problem. Those two possibilities lead to different treatments, in the first case a simple backtrack will lead to another possibility and in the second case, we must start a relaxation process.

Therefore, we would like to collapse the two possibilities into a single one: start the relaxation process in case of contradiction. This can be achieved

by redefining the labeling in terms of adding and suppressing constraints³.

- *Which constraint(s) to suppress ?* Which constraint or which set of constraints do we need to suppress (relax) in order to obtain a satisfiable subproblem. The previous systems (Cousin’s system and IHCS) proposed answers to this question (see section 3.2).

We would like to do it more accurately exploiting all the information that the solver can gather i.e. the justifications for deletions of values (not only trivial justification).

- *How to delete a constraint ?* How can we efficiently perform the suppression of a constraint. An obvious answer is to use systems that can add **and** retract constraints, i.e. incremental systems as [10, 17].

Another answer comes from the CSP community. As we stated before, DnAC4 achieves dynamic arc-consistency and thus enforces the maintenance of arc-consistency after the deletion of a constraint.

A Constraint Relaxation system must provide an efficient answer to the *all* three questions. For example, using DnAC4 to perform the effective suppression does not allow an efficient answer to the *who* question. This is the reason why we would like to perform the deletion easily but keeping in mind that we have to answer other questions. So, we would like to use a DnAC4 like method but with no suppression of information (i.e. not really putting back a value but merely make temporarily unbelievable its deletion).

We propose in the following sections a specialization of the Assumption-based Truth Maintenance Systems concepts to implement a constraint relaxation system which fully answers the three basic questions upon which our framework lies.

4 Truth Maintenance Systems

4.1 Overview of Truth Maintenance Systems

As stated by de Kleer [5], most problem solvers search and are constantly confronted with the obligation to select among equally plausible alternatives. When solving problems, one aims to provide a good answer to two questions:

- How can the search space be efficiently explored, or how can maximum information be transferred from one point in the space to another ?
- How, conceptually, should the problem solver be organized ?

First of all, classical backtracking techniques cannot provide an efficient scheme to achieve such tasks. The problems caused by such approaches lead to the development of truth maintenance systems (TMS). These systems make a clean division between two components: a problem solver which draw inferences

and another component (namely the TMS) solely concerned with the recording of those inferences (called justifications) (see figure 1).

The TMS serves three roles:

- It maintains a cache of all the inferences ever made. Thus inferences, once made, need not be repeated and contradictions, once discovered, are avoided in the future,
- It allows the use of non-monotonic inferences and thus needs a procedure (called truth maintenance) to determine what data are to be believed,
- It ensures that the database is contradiction free. Contradictions are removed by identifying absent justification(s) whose addition to the database would remove the contradiction. This is called dependency directed backtracking.

The Assumption-based Truth Maintenance Systems (ATMS) extend the TMS architecture, by extending the cache idea, simplifying truth maintenance and avoiding dependency directed backtracking.

In a classical (justification-based) TMS every datum is associated with a status of *in* (believed) or *out* (not believed). The entire set of *in* data defines the current context.

In an ATMS each datum is labeled with the sets of assumptions (representing the *environment*) under which it holds. These environments are computed by the ATMS from the problem-solver-supplied justifications.

Those labels enable an efficient way of shifting context. When shifting contexts, one wants to know how much of what was believed in a previous context can be believed in the new one. This can be efficiently achieved by ATMS using the labels and testing their accuracy in the current context.

Chronological backtracking after failure loses useful information because assertions derived from data unaffected by a contradiction remain useful. Thus, only assertions directly affected by the contradiction causing the failure should be removed. In an ATMS, one can tell easily whether an assertion is affected or not.

³ for example, in order to label the variable X whose domain is $[1, 2, 3]$, the first step is to add the constraint $X = 1$, and if this not succeeds then perform: retract $X = 1$ and add $X = 2$.

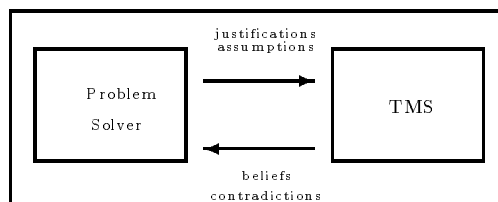


Fig. 1. *TMS and Problem Solver*

4.2 Basic definitions

Here are some basic definitions for ATMS.

- A *node* corresponds to a problem-solver datum.
- An *assumption* designates a decision to assume without any commitment. An assumption is a special kind of node.
- A *justification* describes how a node is derivable from other nodes.
- A *context* is formed by the assumptions of a consistent environment combined with all nodes derivable from those assumptions.
- A *label* is a set of environments associated with every node. A label is consistent i.e. from the label and the set of justifications, one can prove that the current node is to be believed.

While a justification describes how the datum is derived from immediately preceding antecedents, a label environment describes how the datum ultimately depends on assumptions. These sets of assumptions can be computed from the justifications, but computing them each time would disable the efficiency advantage of the ATMS.

The fundamental task of the ATMS is to guarantee that the label of each node is consistent, sound, complete and minimal with respect to the justifications (see [5] for more details).

5 Constraint Relaxation using ATMS

In this section, we show how ATMS can handle uniformly the three components of a Constraint Relaxation system.

5.1 Architecture

The overall architecture of our system is as stated in figure 2. In this figure, the problem solver is a $CLP(\mathcal{FD})$ solver which uses an Arc Consistency algorithm and a labeling procedure coupled with an analyzer which computes constraint(s) to relax from the ATMS information.

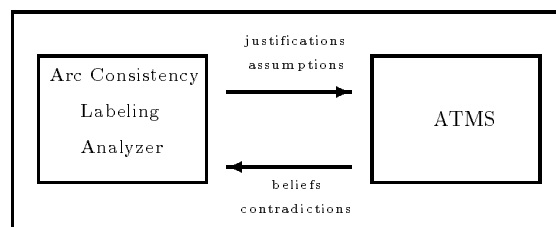


Fig. 2. *ATMS and CLP(FD)*

We give the meaning of the ATMS concepts in our Constraint Relaxation scheme.

- A *node* reflects the suppression of a value in the domain of a variable, which represents the atomic operation of a constraint solver over finite domains. The constraint solver gives a *justification* of this operation. This justification is composed with the *conjunction of constraints* that are responsible for this suppression. This justification is in fact the *environment* explaining the corresponding suppression. This definition of a justification represent the difference between our approach and DnAC4, Cousin’s method or IHCS.
- Each constraint in the problem appears as an *assumption*. This assumption can be *in* or *out* whether the constraint is considered *in* the system or *out* of the system.
- The current context is represented by the *in* assumptions i.e. the active constraints.
- The label of a node is a set of environments which indicates the reason why a value has been retracted.

This architecture is very flexible and allows the use of any constraint solver over Finite Domains. The efficiency our system is related to the precision and accuracy of the justifications provided by the constraint solver. A constraint solver that uses a lot of information to remove a value from a domain will give all this information to our system and thus allow an efficient relaxation procedure.

Our system gives a useful help to the constraint solver but does not create information by itself, it can only use the information contained in the solver.

5.2 Utilization

The ATMS as previously described provides a unified framework to answer the three main questions.

- The identification of responsibility will be given by the node label, which answers the **who** question. As we stated before, the label gives all the conjunctions of constraints that are responsible for deletion. We then have to determine a set of constraints whose relaxation will free at least one value back in the domain.
- Contradictions are raised when the domain of a variable becomes empty. When a contradiction occurs (the domain of a variable, the *failing variable*, becomes empty), the current system of constraints is not satisfiable, we start the determination of constraint(s) to relax. We answer here the **when** question.
- Incrementality will be ensured by keeping useful information between relaxation thanks to the use of good labels. This will answer the **how** question. To relax a constraint, one just have to specify that the related ATMS assumptions may not be believed anymore. This is done by switching the status of the assumption (constraint) from the *in* status to the *out* status.

If the problem solver raises a contradiction during the labeling phase (answering the **when** question), responsibility (answering the **who** question) will efficiently be determined by examining the associated labels. These labels contain all the necessary information to enforce this task.

Once responsibility determined using the previously called analyzer, one just need to retract (answering the **how** question) the corresponding assumption(s) and to reexecute the labeling phase in order to ensure a complete exploration of the new search space⁴.

6 Example

We describe our constraint relaxation scheme and the use of the ATMS considering the Conference problem [4].

6.1 Stating the Conference problem

Michael, John and Alan must attend work-sessions taking place over four half days.

John and Alan want to present their work to Michael and Michael wants to present his work to John and Alan.

Let Ma , Mj , Am , Jm be the four presentations (Michael to Alan, Michael to John, Alan to Michael, John to Michael respectively). Those variables have the domain $[1, 2, 3, 4]$. Each presentation takes one half-day.

Michael wants to know what John and Alan have done before presenting his work. Michael would like not to come the fourth half-day and Michael does not want to present his work to Alan and John at the same time.

Someone who attends a presentation cannot present something in the same half-day. Two different persons cannot present to the same person at the same time.

We introduce the following constraints:

- $C_1 : Ma > Am$, $C_2 : Ma > Jm$, $C_3 : MJ > Am$ and $C_4 : MJ > Jm$. Those constraints are the most important ones so they are given the weight 1.
- $C_5 : Ma \neq 4$, $C_6 : MJ \neq 4$, $C_7 : Am \neq 4$ and $C_8 : Jm \neq 4$. Those constraints are the least important ones, so they are given the weight 3.
- $C_9 : Ma \neq MJ$. This constraint is quite important, so it is given the weight 2.
- $C_{10} : Ma \neq Am$, $C_{11} : Ma \neq Jm$, $C_{12} : MJ \neq Am$, $C_{13} : MJ \neq Jm$ and $C_{14} : Am \neq Jm$. Those constraints are required, so they are not given any weight.

⁴ The search space has been modified by the suppression of a constraint.

Variable	Labels				Resulting Domain
	1	2	3	4	
<i>Ma</i>	[[C_2],[C_1]]	[]	[]	[]	[2,3,4]
<i>Mj</i>	[]	[]	[]	[]	[1,2,3,4]
<i>Am</i>	[]	[]	[]	[[C_1]]	[1,2,3]
<i>Jm</i>	[]	[]	[]	[[C_2]]	[1,2,3]

Table 1. Introduction of the first two constraints

Variable	Labels				Resulting Domain
	1	2	3	4	
<i>Ma</i>	[[C_2],[C_1]]	[]	[]	[[C_5]]	[2,3]
<i>Mj</i>	[[C_4],[C_3]]	[]	[]	[[C_6]]	[2,3]
<i>Am</i>	[]	[]	[[C_3,C_6],[C_1,C_5]]	[[C_7],[C_1],[C_3]]	[1,2]
<i>Jm</i>	[]	[]	[[C_4,C_6],[C_2,C_5]]	[[C_8],[C_2],[C_4]]	[1,2]

Table 2. Introduction of the 14 constraints

6.2 Solving the Conference problem

The ATMS labels are associated with each removal of a value in a domain, so this label can be attached with the corresponding value instead of creating explicitly an ATMS node.

For example, after the introduction of the first two constraints, we obtain the table 1.

For the variable *Ma*, the set of labels can be read as:

- The value 1 has to be removed from the domain if the constraint C_2 is to be believed or if the constraint C_1 is to be believed.
- The values 2, 3 and 4 cannot be removed with the current information⁵.

After the introduction of the 14 constraints, the labels are as in table 2.

For the variable *Am*, the set of labels can be read as:

- The values 1 and 2 cannot be removed with the current information.
- The value 3 has to be removed from the domain if the constraints C_3 and C_6 are both to be believed or if the constraints C_1 and C_5 are both to be believed.
- The value 4 has to be removed from the domain if the constraint C_7 is to be believed or if the constraint C_1 is to be believed or if the constraint C_3 is to be believed.

⁵ We use here arc consistency with AC5 [18]. We can parameterize the real signification of the label with the achieved degree of consistency.

Domain Value	Label
1	[[C_2, C_5, C_{16}], [C_3], [C_4]]
2	[[$C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9$], [C_{16}]]
3	[[$C_4, C_5, C_6, C_9, C_{16}$], [$C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9$]] [C_1, C_5, C_9, C_{16}], [C_2, C_5, C_9, C_{16}]]
4	[[C_6]]

Table 3. *The second contradiction*

Considering the context (the active constraints) we find the current domain of the variable.

To ensure the existence of a solution to this problem, we have to enumerate the variables. This is the labeling phase. We choose to first enumerate the variable Am . Giving the value 1 ($C_{15} : Am = 1$ weight⁶ 100) to Am leads to a contradiction on the variable Mj so we try to give⁷ the value 2 ($C_{16} : Am = 2$ weight 100). This leads to another contradiction for the variable Mj .

Handling the contradictions Contradictions occur when the *current* domain of a variable becomes empty, i.e. all the values have got a non empty label and those labels are valid (i.e. can be believed given the current context).

In order to find the responsible constraint(s) of the a contradiction, we have to *read* the labels for the variable who provoked the contradiction.

In the Conference problem, the labels of the variable Mj after the second contradiction are as in table 3.

We only keep the valid environments (justifications) in these labels. The real labels contained information about the constraint C_{15} which is not valid i.e. relaxed in the current context.

The classical interpretation of the labels leads to following statement:

In order to recover from the inconsistency lying on the variable Mj , we have to free a value in the domain, i.e. we have to make a label invalid by modifying the current context. This modification lies upon the relaxation of a constraint.

In order to recover from the inconsistency, we have to relax:

(C_2 or C_5 or C_{16}) and C_3 and C_4 or (C_1 or ... or C_9 and ... and C_{16}) or ... or C_6

This formulation is not satisfactory because, we have to choose the value that we want to make free. In order to avoid this choice, we transform the preceding disjunction of conjunctions in a conjunction of disjunctions, i.e. :

⁶ This is an arbitrary great value.

⁷ We must first relax the constraint C_{15} .

$(C_1 \text{ or } \dots \text{ or } C_9 \text{ or } C_{16}) \text{ and } (C_1 \text{ or } \dots \text{ or } C_9)$
and \dots **and** $(C_4 \text{ or } C_5 \text{ or } C_6 \text{ or } C_9 \text{ or } C_{16})$

We then have to make sure that in each conjunction there is at most one true component considering that C_i is true if the constraint is relaxed and false otherwise.

This problem can be seen as the determination of a set covering problem in an hypergraph whose vertices are the different disjunctions and whose edges are the different constraints. An edge represents the sharing of a constraint between different disjunctions. An edge can involve only one disjunction. This set covering must minimize a certain criterion. This criterion defines the nature of the solutions we want to determine to solve our Constraint Relaxation problem.

A criterion could be:

- minimizing the *cost* of the relaxation, i.e. to minimize the sum of the weights of the edges. In our case, the weight would have been the inverse of the weight associated to the constraint. This problem is \mathcal{NP} -Complete. In our example, this method leads to the set of relaxable constraints: $\{C_5\}$.
- minimizing the maximum weight of the covering, i.e. ensuring that the more important of the relaxed constraint is the least important one. This can be answered by simply selecting the least important constraint in each disjunction, i.e. selecting for each vertex the better edge (the one with the smaller weight). This can be done in a linear time: $O(m)$ where m represents the size of the formula. In our example, this method leads to the set of relaxable constraints: $\{C_5, C_{16}\}$.

Performing the relaxation We simply have to remove from the context (change their status from *in* to *out*, which is immediate) the constraints determined in the previous step and then reexecute the labeling procedure. As we reexecute the labeling procedure we benefit from the previously done work because the marks did not changed during the relaxation (we did not erase any deduction).

Giving an answer If the labeling procedure can be achieved, the current solution is our solution, and the satisfied constraints are those in the current context. In our example, we reconsider the value 1 for the variable Am which leads to a solution $[Ma, Mj, Am, Jm] = [4, 3, 1, 2]$ which violates only one constraint: C_5 .

7 First results

We implemented our system in Prolog. We chose the AC5 algorithm [18] to achieve arc-consistency. The complete system represents 3000 lines of code.

System	Relaxed Constraint Backtracks	
HCLP	1	27
IHCS	3	3
FELIAC	1	3
AC+ATMS	1	2

Table 4. Complete Results for the Conference Problem

7.1 Comparing systems on the Conference Problem

We chose to compare our system with the more complete (answering the three basic questions) systems we found in the literature. We did not compare it with incremental algorithms (DnAC*) since the more important basic questions are *how and who*.

So, we implemented HCLP and IHCS as presented in the related papers to make comparisons between different systems. We also used the FELIAC system presented in [2]. This system uses known techniques (IHCS, Van Hentenryck’s oracles [17]) to create a constraint relaxation system fully but not uniformly answering the three basic questions. This system is interactive, the user must give the constraint to add or retract (FELIAC gives a choice of judicious constraints), it cannot deal with a Prolog program.

We can find the results on the Conference problem for the four systems in table 4.

To obtain a solution in the Conference problem, one needs to relax only one constraint (namely C_5). The HCLP and FELIAC systems gave the *good* answer as well as our system (AC+ATMS). IHCS had to relax three constraints to obtain a feasible solution. Let us recall that IHCS works on the failing *constraint* and not the failing *variable* and so works with incomplete information, that’s why it has to relax inaccurate constraints.

When we look at the number of backtracks which reflects the real efficiency of the systems, we can see that HCLP generates a lot of backtracks compared to the other systems. This illustrates the inaccuracy of such a method with incomplete solvers, after each addition of variable, HCLP must ensure that there exist a solution and so has to exhibit one when using arc-consistency.

Finally, our system is more efficient than FELIAC and illustrates the incremental capability of ATMS. In other words, the cache of inferences of the ATMS is more efficient than Van Hentenryck’s oracles.

7.2 Complexity

In this section, we give first results about complexity in the worst case of our implemented system. We cannot really compare the complexity of our system with other systems since we propose a complete treatment of the relaxation

problem (answering the three basic questions) whereas no other system, as far as we know, proposes it.

Let ϵ the number of constraints in the problem, n the number of variables and d the size of the largest domain. We give here the final results for complexity using AC5 as constraint solver.

Spatial Complexity The space use of our system will increase when adding justifications. We cannot had more justifications than the solver removes values.

A justification can contain at most ϵ constraints. Adding a justification depends on the constraint solver. When using AC5, at each step of the algorithm, a justification can be added to explain the removal of a value in a domain for a variable. Thus, the space complexity would be $O(nd \times \epsilon \times ed)$ in the worst case since there are nd labels in the system. This complexity is considered without the ATMS treatment which ensure the minimal size of the label by suppressing redundant justifications in a same label.

For the Conference problem, the number of added justifications would be, in the worst case, $O(ned^2) = O(4 \times 14 \times 4^2) = O(896)$. The final number of justifications for the Conference problem is 61.

Time complexity The time complexity of the constraint solver is not affected by the use of the ATMS system although at each removal the constraint solver must justify his removal to the ATMS recording system (this operation is not significant).

The set of labels for a variable is transformed in a normal conjunctive form at each relaxation. This transformation is achieved in $O(m^2)$ where m is the size of the treated set. This set would contain in the worst case all the added justifications : $O(ned^2)$. For the Conference problem, as stated in table 3 this set contains only 10 justifications.

The selection of the constraints to relax, (see section 6.2) depends on the retained criterion. When using the second criterion, the complexity is: $O(m)$, of course when using the first criterion we try to solve an \mathcal{NP} -complete problem and therefore the complexity does not remain polynomial.

8 Conclusion and further works

Using ATMS for constraint relaxation provides a uniform framework to answer the three basic questions (when, which and how). This framework is very flexible and can be used with any constraint solver (domain reduction) to provide an automated constraint relaxation system over finite domains.

The IHCS system and DnAC4 were only focused in a single aspect of the problem. Our results enlight the utility of tackling the three questions in the same framework.

Some further works are to be done:

- First, to evaluate the average behavior of our approach, in particular in terms of space complexity. We plan to study how to retain only useful information when adding a justification.
- Our research area lacks for significant benchmarks. We plan to study how to generate random instances of intractable problems avoiding simple contradictions.
- De Kleer [6] shows how CSP can be mapped to ATMS by encoding each domain and constraint as boolean formulas, and then achieving arc-consistency by defining inference rules. It would be interesting to extend this scheme to Dynamic CSP, giving a semantics for constraint deletion.
- Saraswat *et al.* [15] initiated works to actively use the ATMS labels as boolean formulas. This leads to the integration of those formulas inside the constraint language. This interesting approach would worth a study for finite domains.

References

1. Christian Bessière. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*, 1991.
2. Patrice Boizumault, Christelle Guéret, and Narendra Jussien. Efficient labeling and constraint relaxation for solving time tabling problems. In Pierre Lim and Jean Jourdan, editors, *Proceedings of the 1994 ILPS post-conference workshop on Constraint Languages/Systems and their use in Problem Modeling : Volume 1 (Applications and Modelling)*, Technical Report ECRC-94-38, ECRC, Munich, Germany, November 1994.
3. Alan Borning, Michael Maher, Amy Martindale, and Molly Wilson. Constraint hierarchies and logic programming. In Giorgio Levi and Maurizio Martelli, editors, *ICLP'89: Proceedings 6th International Conference on Logic Programming*, pages 149–164, Lisbon, Portugal, June 1989. MIT Press.
4. Xavier Cousin. Meilleures solutions en programmation logique. In *Proceedings Avignon'91*, 1991. In French.
5. Johan de Kleer. An assumption-based tms. *Artificial Intelligence*, 28:127–162, 1986.
6. Johan de Kleer. A comparison of ATMS and CSP techniques. In *IJCAI-89: Proceedings 11th International Joint Conference on Artificial Intelligence*, pages 290–296, Detroit, 1989.
7. Romuald Debruyne. DnAc6. Research Report 94-054, Laboratoire d'Informatique, de Robotique et de Micro-électronique de Montpellier, 1994. In French.
8. François Fages, Julian Fowler, and Thierry Sola. Handling preferences in constraint logic programming with relational optimization. In *PLILP'94*, Madrid, September 1994.
9. François Fages, Julian Fowler, and Thierry Sola. A reactive constraint logic programming scheme. In *International Conference of Logic Programming, ICLP'95*, Tokyo, 1995.
10. Bjorn Freeman-Benson, John Maloney, and Alan Borning. An incremental constraint solver. *Communications of the ACM*, 33(1):54–63, January 1990.

11. Eugene Freuder. Partial constraint satisfaction. In *IJCAI-89: Proceedings 11th International Joint Conference on Artificial Intelligence*, pages 278–283, Detroit, 1989.
12. Alois Haselböck, Thomas Havelka, and Markus Stumptner. Revising inconsistent variable assignments in constraint satisfaction problems. In Manfred Meyer, editor, *Constraint Processing: Proceedings of the International Workshop at CSAM'93, St. Petersburg, July 1993*, Research Report RR-93-39, pages 113–122, DFKI Kaiserslautern, August 1993.
13. Michael Jampel and David Gilbert. Fair Hierarchical Constraint Logic Programming. In Manfred Meyer, editor, *Proceedings ECAI'94 Workshop on Constraint Processing*, Amsterdam, August 1994.
14. Francisco Menezes, Pedro Barahona, and Philippe Codognet. An incremental hierarchical constraint solver. In Paris Kanellakis, Jean-Louis Lassez, and Vijay Saraswat, editors, *PPCP'93: First Workshop on Principles and Practice of Constraint Programming*, Providence RI, 1993.
15. Vijay Saraswat, Johan de Kleer, and Brian Williams. ATMS-based constraint programming. Technical report, Xerox PARC, October 1991.
16. Gilles Trombettoni. CCMA*: A Complete Constraint Maintenance Algorithm Using Constraint Programming. In Manfred Meyer, editor, *Constraint Processing: Proceedings of the International Workshop at CSAM'93, St. Petersburg, July 1993*, Research Report RR-93-39, pages 123–132, DFKI Kaiserslautern, August 1993.
17. Pascal Van Hentenryck. Incremental constraint satisfaction in logic programming. In *Proceedings 6th International Conference on Logic Programming*, 1989.
18. Pascal Van Hentenryck, Yves Deville, and Choh-Man Teng. A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57(2–3):291–321, October 1992.