NMUS: Structural Analysis for Improving the Derivation of All MUSes in Overconstrained Numeric CSPs

R.M. Gasca, C. Del Valle, M.T. Gómez-López, and R. Ceballos

Departmento de Lenguajes y Sistemas Informáticos. Escuela Técnica Superior de Ingeniería Informática. Universidad de Sevilla. (Spain)
{gasca,carmelo,mayte,ceballos}@lsi.us.es

Abstract. Models are used in science and engineering for experimentation, analysis, model-based diagnosis, design and planning/sheduling applications. Many of these models are overconstrained Numeric Constraint Satisfaction Problems ($NCSP$), where the numeric constraints could have linear or polynomial relations. In practical scenarios, it is very useful to know which parts of the overconstrained NCSP instances cause the unsolvability.

Although there are algorithms to find all optimal solutions for this problem, they are computationally expensive, and hence may not be applicable to large and real-world problems. Our objective is to improve the performance of these algorithms for numeric domains using structural analysis. We provide experimental results showing that the use of the different strategies proposed leads to a substantially improved performance and it facilitates the application of solving larger and more realistic problems.

1 Introduction

A lot of Artificial Intelligence problems can be cast in terms of Numeric Constraint Satisfaction Problems ($NCSPs$), and a large number of systems have been developed to compute efficiently solutions of these problems. $NCSPs$ are more and more often used to solve engineering problems arisen in different areas such as qualitative reasoning, diagnosis, planning, scheduling, configuration, distributed artificial intelligence, etc... This work focuses on problems related to engineering field, what play a prominent role in industrial applications. Generally, these problems are formed by a set of constraints among variables whose domains are real interval values. Usually, the numeric constraints are linear or polynomial relations (equations or inequations).

However, not every set of numeric constraints is satisfiable. Different researchers have proposed methods for the identification of Minimally Unsatisfiable Subsets of Constraints ($MUSes$) or Conflict Sets (CS) as they are also named in overconstrained $CSPs$. Determining $MUSes$ can be very valuable in many industrial applications, because it describes what is wrong in a $NCSP$

instance. They represent the smallest explanations -in terms of the number of involved constraints- of infeasibility. Indeed, when we check the consistency of a $NCSP$, we prefer knowing which constraints are contradicting one another rather than only knowing that the whole $NCSP$ is inconsistent.

In the bibliography, different types of $CSPs$ have been treated in order to obtain the $MUSes$. They are related to Satisfiability Problems [8] [2] [13] [7], Disjunctive Temporal Problem (DTP) [11] [9] [12] and model-based diagnosis and debugging problems [10] [5] [6] [1] [3]. Due to the high computational complexity of these problems, the goal of most of these approaches was to reduce the amount of satisfaction checking and subsets examined. However, some approaches were designed to derive only some $MUSes$ and no all $MUSes$ of these overconstrained $CSPs$.

To derive $MUSes$ in overconstrained $NCSP$, we are aware of very few technical works. In [4], Irreducible Infeasible Subsets (IIS) was studied for only linear and integer domains, but not all MUSes are obtained. These problems may contain multiple MUSes, and all of them must be resolved by constraint relaxation before the $NCSP$ can be solved. Also, other authors of the model-based diagnosis community have treated the high complexity of these problems using constraint databases [6] and new concepts such as constraint clusters and nodes [3].

In this paper, a set of new derivation techniques are presented to obtain efficiently $MUSes$ of a overconstrainted $NCSP$. These techniques improve the complete technique in several ways depending on the structure of the constraint network. It makes use of the powerful concept of the structural lattice of the constraints and neighborhood-based structural analysis to boost the efficiency of the exhaustive algorithms. As systematic methods for solving hard combinatorial problems are too expensive, structural analysis offers an alternative approach for quickly generating all $MUSes$. Accordingly, experimental studies of these new techniques outperform the best exhaustive ones. They avoid to solve a high number of $NCSPs$ with exponential complexity, however they add some new procedures with polynomial complexity.

The rest of the article is organized as follows. In Section 2, we start presenting some examples of overconstrained $NCSPs$ to introduce the problem domain. Section 3 presents some definitions and notations. Section 4 exposes different neighborhood concepts based on the structural analysis of the constraint network. Afterwards, we propose different search algorithms for deriving numeric $MUSes$ in a efficient way and their experimental results are argued in Section 5. Finally, in the last section we present our conclusions and future work.

2 Motivating Examples

The parts of an overconstrained $NCSP$ instance that could cause the unsolvability are the variables domains or constraints of the problem. Only this last cause will be treated in this article.

In the following subsections, we specify some different $NCSP$ instances to motivate this work. The specification of a $NCSP$ instance is represented by Ψ,

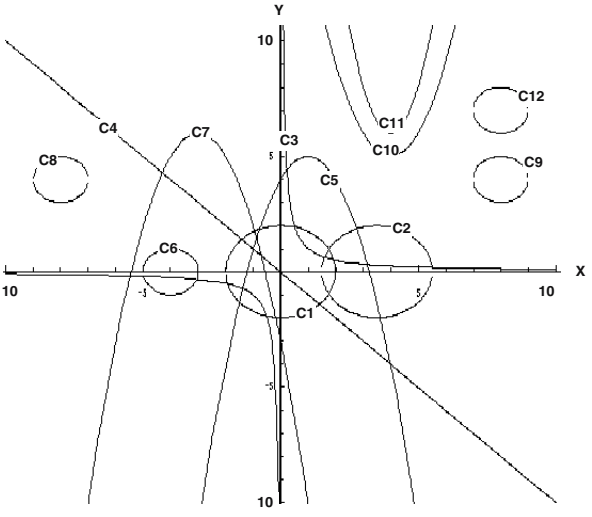

Fig. 1. Overconstrained NCSP with same real variables for all constraints

the variables by X^{Ψ}, the domains by D^{Ψ}, the constraints by C^{Ψ} and the goals by G^{Ψ}. In this last item, the modeler could also specify which constraints would be preferred for relaxing.

2.1 *NCSP* with the Same Real Variables for All Constraints

An example is the following geometrical problem, where the overconstrained $NCSP$ instance has linear equations and polynomial equations/inequations:

$$\Psi \equiv \begin{cases} X^{\Psi} = \{x, y\} \\ D^{\Psi} = \{x, y \in [-10, +10]\}, \\ C^{\Psi} = \{c_1 \equiv x^2 + y^2 < 4, c_2 \equiv\ (x - 7/2)^2 + y^2 < 4, c_3 \equiv x * y > 1, \\ \qquad c_4 \equiv\ x + y = 0, c_5 \equiv\ y + (x-1)^2 = 5,\ c_6 \equiv (x+4)^2 + y^2 = 1, \\ \qquad c_7 \equiv\ y = 6 - (x+3)^2, c_8 \equiv (x+8)^2 + (y-4)^2 = 1, \\ \qquad c_9 \equiv\ (x-8)^2 + (y-4)^2 = 1, c_{10} \equiv y = 5 + (x-4)^2, \\ \qquad c_{11} \equiv y = 6 + 2 * (x-4)^2, c_{12} \equiv (x-8)^2 + (y-7)^2 = 1\} \\ G^{\Psi} = Solutions(X)? \quad Why? \end{cases}$$

This problem has no solution, but the question is what cause it. In this case, Ψ exhibits the following MUSes, namely $\{c_1, c_2, c_5\}$, $\{c_{10}, c_{11}\}$, $\{c_9, c_{12}\}$, etc...

2.2 *NCSP* with Some Different Variables for the Numeric Constraints

The following example is extracted from a recent work in the model-based diagnosis community [3], where the m_i and a_i constraints corresponds to multipliers and adders respectively. This is a very illustrative example to show the utility of the structural analysis:

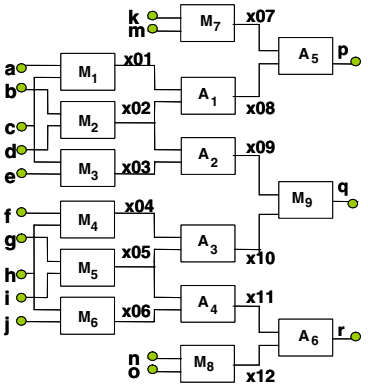

Fig. 2. Overconstrained NCSP with different real variables for all constraints

$$\Psi \equiv \begin{cases} X^{\Psi} = \{a, b, c, d, e, f, g, h, i, j, k, m, n, o, p, q, r, x_i (i \in \{1, ..., 12\})\} \\ D^{\Psi} = \{x_i \in (-\infty, +\infty), \\ \quad a, b, c, d, e, f, g \in [2, 4] \;\; k, m, n, o \in [3, 5] \\ \quad p, q, r \in [15, 18])\} \\ C^{\Psi} = \{m_1 \equiv a * \; c \; = x_{01}, \; a_1 \equiv \; x_{01} + x_{02} = x_{08}, \\ \quad etc...\} \\ G^{\Psi} = \; Solutions(X)? \quad Why? \end{cases}$$

3 Definitions and Notations

In the previous section, some overconstrained $NCSPs$ in terms of examples have been shown. This section presents some necessary definitions to formalize and clarify the derivation of all $MUSes$ in these problems.

Definition 3.1 (Numeric Variable). A variable of the $NCSP$ whose domain is a real interval value. The set of numeric variables of the problem is denoted by X^{Ψ} and $X^{\Psi}(c_i)$ stands for the set of variables of a constraint c_i.

Definition 3.2 (Numeric Constraint). It is a linear or polynomial relation (equations or inequations) involving a finite subset of numeric variables.

Definition 3.3 (Goal). A predicate that denotes the users' preferences to search why the $NCSP$ is overconstrained.

Definition 3.4 (Numeric Constraint Satisfaction Problem). A four-tuple $\Psi = (X,D,C,G)$ where $X^{\Psi} = \{x_1, ..., x_n\}$ is a set of variables, whose continuous domains are respectively $D^{\Psi} = \{d_1, ..., d_n\} (n \geq 1)$, $C^{\Psi} = \{c_1, ..., c_m\} (m \geq 1)$ is a set of numeric constraints and G^{Ψ} is the goal.

Definition 3.5 (Overconstrained NCSP). It is a $NCSP$ with no solution caused by some of the domains or constraints contradicting others.

When a $NCSP$ instance is unsatisfiable, it has at least one *Numeric Minimally Unsatisfiable SubSet*, in short one $NMUS$. It is a set of numeric constraints which is unsatisfiable, but becomes satisfiable as soon as we remove any of its constraints.

Definition 3.6 (Numeric Minimally Unsatisfiable SubSet). Given an instance Ψ of a $NCSP$, a MUS μ is a set of numeric constraints s.t. $\mu \subset C^{\Psi}$, μ is unsatisfiable and $\forall \delta \in \mu$, $\mu \backslash \{\delta\}$ is satisfiable.

The number of constraints in a MUS is its cardinality and it is represented by $\#\mu$. Generally, we can have more than one MUS in the same $NCSP$. Some of them can overlap, in the sense that they can share some constraints, but they cannot be fully contained one in another. This concept of MUS have similarities with that one of IIS in the case of systems of linear inequalities.

4 Neighborhood-Based Structural Analysis

To construct a practical system developing a complete and efficient method for deriving numeric $MUSes$ is a key issue in real-world and industrial applications. In this paper, the option for an important reduction of the search space is based on the structural analysis. For this reason, the concept of neighbors of a given assignment of the constraints of a $NCSP$ is defined.

Definition 4 (Assignment). It is the tuple of values $\{0, 1\}$ assigned to each constraint of a $NCSP$ instance Ψ, meaning the truth value of each constraint. It is represented by $A(C^{\Psi})$.

For example, in a $NCSP$ with five numeric constraints an assignment could be $(c_1, c_2, c_3, c_4, c_5) \equiv (1, 0, 0, 1, 0)$.

4.1 General Neighborhood

Definition 4.1 (Neighbor Assignment). Given an assignment $A(C^{\Psi})$, a neighbor assignment is defined by a new assignment $A'(C^{\Psi})$ that differs in exactly one truth value.

For a given assignment $A(C^{\Psi})$, one option could be to consider all alternatives whose variable assignments differ in exactly one position; for example, the assignment $(c_1, c_2, c_3, c_4, c_5) \equiv (1, 0, 0, 1, 0)$ would be a neighbor of $(c_1, c_2, c_3, c_4, c_5)$ $\equiv (1, 0, 0, 1, 1)$, since they both differ only on the assignment to c_5. However, as each c_i variable may take these different values, the cardinality of the set of possible neighbors could be very high. But, it could be reduced in a significant way, taking into account the structural aspects of the constraint network of the $NCSP$. In this article, two clear options of neighborhood are used: when the variables of all the constraints of $NCSP$ are identical then we define the concept of domain-based neighborhood; in other cases we define the concept of variable-based neighborhood.

4.2 Variable-Based Neighborhood

An important aspect in this concept is the notion of *Non-Observable Numeric Variable* of a $NCSP$ instance. For these variables, there is not any information about their domains.

Definition 4.2.1 (Non-Observable Numeric Variable). It is a numeric variable of a $NCSP$, whose initial domain in the problem specification is $(-\infty, +\infty)$.

For the example in the subsection 2.2, the neighborhood is based on the common *non-observable numeric variables* between constraints.

Definition 4.2.2 (Variable-based Neighbor Assignment of Constraints). Given an assignment of $A(C^{\Psi})$, a neighbor is defined by a new assignment $A'(C^{\Psi})$ that differs in exactly one truth value of a constraint with some common non-observable variable to the constraints with truth values equals 1.

4.3 Domain-Based Neighborhood

Another neighborhood concept is when all numeric constraints of an overconstrained $NCSP$ instance have the same variables. In this case we could use the projection operator of a variable $x_i \in X^{\Psi}$ w.r.t. a constraint $c_j \in C^{\Psi}$ is represented as $\Pi_{x_i}(c_j)$. In the same way, the projection operator of a variable $x_i \in X^{\Psi}$ w.r.t a set of constraint $C^{\Gamma} \subset C^{\Psi}$ that is represented as $\Pi_{x_i}(C^{\gamma})$. Then, the new concept for deriving $MUSes$ is the domain-based neighborhood. A constraint $c_i \in C^{\Psi}$ could be domain-based neighbor of another set of constraint $C^{\gamma} \subset C^{\Psi} | c_i \notin C^{\gamma}$ when the intersection of the projections for all variables of set X^{Ψ} is not empty:

$$\forall x_k \in X^{\Psi} \quad \Pi_{x_k}(c_i) \cap \Pi_{x_k}(C^{\gamma}) \neq \emptyset$$

Definition 4.3 Domain-based Neighbor Assignment of the Constraints Given an assignment of $A(C^{\Psi})$, a neighbor is defined by a new assignment $A'(C^{\Psi})$ that differs in exactly one truth value of a constraint and all projection operations of the variables w.r.t. a set of the numeric constraints with truth value equals 1 is not empty.

With this definition we are sure of the domain-based neighborhood, but it could happen that the intersection of all the projections are not empty and the constraints are unsatisfiable. For this reason, it is necessary to solve a $NCSP$.

5 NMUS: Numeric MUSes Search Methods

In this section, a set of methods $NMUS$ is presented to efficiently derive all $MUSes$ using the Neighborhood-based Structural Analysis on overconstrained $NCSP$. We describe different bottom-up derivation strategies taking into account the concept of neighborhood for the different types of problems. The search methods are different depending on the structural aspects of these problems.

5.1 NMUS for NCSPs with the Same Variables for All Constraints

A basic algorithm would study all the $2^n - 1$ combinations of constraints in order to determine all $MUSes$ of an overconstrained $NCSP$ instance Ψ, where n is the cardinality of the set C^{Ψ}. The proposed method is complete, but it is very inefficient and no practical. For this reason, this work proposes different strategies to improve this algorithm.

Let $\mathcal{MUS}$ be a data structure List of Sets where the $MUSes$ are stored and $\mathcal{Q}$ a data structure type Queue of Sets where the active constraints and its projections w.r.t the variables of the problem are stored. The function *poll()* retrieves and removes the head of a queue or null if the queue is empty.

First Improvement (NMUS-1): Only Inclusion in queue of satisfiable subsets. This first improvement will include in the queue Q only subset of constraints that are satisfiable. Given an overconstrained $NCSP$ instance Ψ, the algorithm is shown in Algorithm 1.

Alg. NMUS-1 ($\Psi : NCSP$)
Let $C^{\Psi} = \{c_1, \ldots, c_n\}$ be constraints of the overconstrained NCSP instance
$\mathcal{Q}$:= Queue with a set for each satisfiable numeric constraints belong to C^{Ψ}
$\mathcal{MUS}$:= List with the set of unsatisfiable numeric constraints belong to C^{Ψ}
while ($\mathcal{Q}$ is not Empty)
  $\{c_i \ \ldots \ c_j\}$:=$\mathcal{Q}$.poll()
  for ($c_k \in \{c_{j+1}$ to $c_n\}$)
    if (NOT $\exists SubSet^{1\ldots n-1}_{\{c_i \ldots c_j\}} \cup c_k \in \mathcal{MUS}$) // n is cardinality of $\{c_i \ldots c_j\}$
      if ($\{c_i \ \ldots \ c_j\} \cup c_k$ is satisfiable) // a $NCSP$ must be solved
        $\mathcal{Q}$.add($\{c_i \ \ldots \ c_j\} \cup c_k$)
      else
        $\mathcal{MUS}$.add($\{c_i \ \ldots \ c_j\} \cup c_k$)
      endIf
    endIf
  endFor
endWhile

Algorithm 1. NMUS-1 ($\Psi : NCSP$)

In this algorithm, the neighborhood concept is not taken into account and the satisfiability could be checked using $NCSP$ solvers.

Second Improvement (NMUS-2): Using Domain-based neighborhood In this algorithm, two concepts are used: domain-based neighborhood and overlapping projection. The initialization procedure is the same as the previous algorithm. The new algorithm is shown in Algorithm 2.

The function **Overlap_Projection**(*Constraint c, Constraints List lc*) returns true if it exists overlapping between the projection of the constraint c and the projection of lc w.r.t. every variable. If this function returns false, it means that it exists a MUS formed by c and some constraints of lc, thereby $c \cup lc$ is not a MUS.

Alg. NMUS-2($\Psi : NCSP$)
..... // Initialization
while ($\mathcal{Q}$ no Empty)
$\{c_i \ldots c_j\}$:= $\mathcal{Q}$.poll() // a list of satisfiable constraints are obtained
for (c_k in $\{c_{j+1}$ to $c_n\}$) // it avoids to obtain redundant solutions
if (Overlap_Projection(c_k, $\{c_i \ldots c_j\}$) AND
NOT $\exists SubSet^{1\ldots n-1}_{\{c_i \ldots c_j\}} \cup c_k \in \mathcal{MUS}$
if ($\{c_i \ldots c_j\} \cup c_k$ is satisfiable) // a $NCSP$ is created
$\mathcal{Q}$.add($\{c_i \ldots c_j\} \cup c_k$)
else
$\mathcal{MUS}$.add($\{c_i \ldots c_j\} \cup c_k$)
endIf
else
if(#$\{c_i \ldots c_j\}$=1)
$\mathcal{MUS}$.add($\{c_i \ldots c_j\} \cup c_k$)
endIf
endIf
endFor
endWhile

Algorithm 2. NMUS-2 ($\Psi : NCSP$)

Third Improvement (NMUS-3): Sorting constraints according to the overlapped domain. The heuristic used in this algorithm is based on the quick search of $MUSes$. First of all, the algorithm sorts the constraints depending on the number of projections that intersect with the projections of other constraints.

It is possible to check the satisfiability only analysing the minimum and maximum value of each variable in the different constraints, no being necessary to solve a $NCSP$. The previous algorithms add subsets of constraints in the queue $\mathcal{Q}$ when a subset of constraints is satisfiable. If we analyze first the less promising subsets, there will be less possibilites yo add these constraints to $\mathcal{Q}$.

5.2 NMUS for NCSPs with Some Different Variables for the Numeric Constraints

In this algorithm, we will apply a different neighborhood concept, the variable-based one. The initialization procedure is the same as in the previous algorithm, but the data structure $\mathcal{Q}$ can be now a Queue, a Stack or another data structure depending on the different search strategy. This structure must have a new method *add* which includes a tuple $\langle C^\gamma, NOBV(C^\gamma)\rangle$, where $NOBV(C^\gamma))$ represents the set of non-observable variables of C^γ.

Depending on the type of structure $\mathcal{Q}$, the search process will be depth-search or breadth-search, what will determine two different algorithms **NMUS-4** and **NMUS-5** respectively (Algorithm 3).

Alg. NMUS-4-5($\Psi : NCSP$)
..... // Initialization
while($\mathcal{Q}$ is not Empty)
$\langle C^\gamma, NOBV(C^\gamma)\rangle := \mathcal{Q}$.poll() // chose a element belong to Q
neighbors := expand ($\langle C^\gamma, NOBV(C^\gamma)\rangle$) // generate neighbours
according variable-based neighborhood
foreach ($\langle c_k, NOBV(c_k)\rangle \in$ neighbors)
if($C^\gamma \cup c_k$ is satisfiable)// a $NCSP$ is created
$\mathcal{Q}$.add($\langle C^\gamma, NOBV(C^\gamma)\rangle \cup \langle c_k, NOBV(c_k)\rangle$)
else
$\mathcal{MUS}$.add($\langle C^\gamma, NOBV(C^\gamma)\rangle \cup \langle c_k, NOBV(c_k)\rangle$)
endIf
endFor
endWhile

Algorithm 3. NMUS-4-5($\Psi : NCSP$)

6 Experimental Results

NMUS is a prototype that includes all previous algorithms. This prototype is implemented in Java and runs on an AMD Athlon Dual Core 2.21 GHz with 1.78 GB Ram. The standard routine used for solving $NCSP$ belongs to $ILOG^{TM} JSolver$.

The different algorithms of this prototype improve the performance of basic algorithms for numeric domains using the structural analysis. We provide experimental results showing that the use of the different strategies proposed leads to substantially improved performance and facilitates to solve larger and more realistic problems. The following table reports the experimental results for the different examples of the Section 2 when a domain-based or a variable-based neighborhood are used. NMUS-5 is more efficient than NMUS-4 since using a breadth search approach we can detect more easily the redundant sets of constraints that are generated. The examples show also a significant improvement w.r.t. the basic algorithms. Therefore these algorithms provide a realistic method for deriving all numeric $MUSes$ of a given problem.

Table 1. Experimental Results for examples in Section 2

Algorithms	*Example 2.1*		*Algorithms*	*Example 2.2*
	# NCSPs	Time(ms)		Time (ms)
Basic Alg.	$2^{12} - 1 = 4095$	40210	Basic Alg.	1017
NMUS-1	88	8692	**NMUS-4**	16,8
NMUS-2	58	7500	**NMUS-5**	2,0
NMUS-3	57	2340		

7 Conclusions and Future Work

The derivation of all $MUSes$ for overconstrained $NCSP$ is a computationally hard problem and arises in a lot of industrial problems. This problem has been formally defined in this paper and different methods for deriving all $NMUSes$

are also presented here. Our experimental results show that the computation time required is significantly reduced in comparison to the basic algorithms.

Future work in this problem will include moreover enhancing more the efficiency of our algorithms, the treatment of new types of problems, for example when the constraint network has cycles or a disjunctive set of constraints. Finally, an important future goal will be to use the relaxation preferences that provides a user about how to weaken constraints to achieve feasibility.

Acknowledgements

This work has been partially supported by the Spanish *Ministerio de Educación y Ciencia* through a coordinated research project(grant DIP2006-15476-C02-01) and Feder (ERDF).

References

1. Bailey, J., Stuckey, P.J.: Discovery of Minimal Unsatisfiable Subsets of Constraints using Hitting set dualization. In: Hermenegildo, M.V., Cabeza, D. (eds.) Practical Aspects of Declarative Languages. LNCS, vol. 3350, pp. 174–186. Springer, Heidelberg (2005)
2. Bruni, R.: Approximating minimal unsatisfiable subformulae by means of adaptive core search. Discrete Applied Mathematics 130, 85–100 (2003)
3. Ceballos, R., Gómez-López, M.T., Gasca, M.T.R.M., del Valle, C.: Integración de técnicas basadas en modelos para la determinación de la diagnosis mínima de un sistema. Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial No. 31, pp. 41–51 (2006)
4. Chinneck, J., Dravnieks, E.: Locating minimal infeasible constraint sets in linear programs. ORSA Journal on Computing 3, 157–168 (1991)
5. de la Banda, M.G., Stuckey, P.J., Wazny, J.: Finding all minimal unsatisfiable subsets. In: PPDP 2003. Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming, pp. 32–43. ACM Press, New York (2003)
6. Gómez, M.T., Ceballos, R., Gasca, R.M., Del Valle, C.: Constraint Databases Technology for Polynomial Models Diagnosis. In: Proceedings DX 2004 (2004)
7. Grégoire, É., Mazure, B., Piette, C.: Local-Search Extraction of MUSes. Constraints 12(3) (2007)
8. Junker, U.: QuickXPlain. In: Conflict Detection for Arbitatrary Constraint Propagation Algorithms Proceedings IJCAI 2001 (2001)
9. Liffiton, M., Sakallah, K.: On finding all minimally unsatisfiable subformulas. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 173–186. Springer, Heidelberg (2005)
10. Mauss, J., Tatar, M.: Computing Minimal Conflicts for Rich Constraint Languages. In: ECAI, pp. 151–155 (2002)
11. Moffitt, M.D., Pollack, M.E.: Applying Local Search to Disjunctive Temporal Problems. In: Proced. IJCAI (2005)
12. Liffiton, M.H., Moffitt, M.D., Pollack, M.E., Sakallah, K.A.: Identifying Conflicts in Overconstrained Temporal Problems. In: Proceedings IJCAI 2007 (2007)
13. Oh, Y., Mneimneh, M.N., Andraus, Z.S., Sakallah, K.A., Markov, I.L.: AMUSE: A Minimally-Unsatisfiable Subformula Extractor. In: Proceedings of the Design Automation Conference (DAC 2004), ACM/IEEE, pp. 518–523 (2004)