

A Best First approach for solving over-constrained dynamic problems

Narendra Jussien and Patrice Boizumault
École des Mines de Nantes

Abstract

Many real-life problems tackled using Constraint Programming are dynamic. Addition and deletion of constraints may lead to inconsistencies. When a solution is required, constraints may be violated in order to fulfill the user requirements. In this paper, we propose a best-first search method (parameterized by the domain of constraints and the associated solver) for solving over-constrained problems arising in dynamic environments. We propose a specialization of this approach for CSP using arc-consistency. We address complexity issues to show the tractability of our approach and describe our implementation: the DECORUM system.

1 Introduction

Many real-life problems are tackled using Constraint Programming. In many applications, modifications of the problems must be integrated because of external modifications (consider timetabling problems). Such dynamic evolutions can generate inconsistent constraint systems. Nevertheless, it may happen that the user requires a solution that satisfies the best as possible the constraints of the problems. In order to fulfill the user's requirements, constraints of the problem need to be violated (relaxed).

The problem is to determine constraints to relax which is the aim of a constraint relaxation system (an exploration method upon the configuration space of the problem). A configuration is a split of the constraints of the problem in two sets: active and relaxed ones. A user defined comparator is used to choose between configurations. An over-constrained problem solver (a constraint relaxation system coupled with a constraint solver) is characterized by the search method used on the configuration space.

In this paper, we propose a best first search method on the configuration space that is well suited for dynamic problems. This search method is parameterized with a domain on which constraints are defined and an associated solver. We present our best-first search method and show that it can be efficiently specialized for CSP using a Deduction Maintenance System. Then, we detail the resulting complexity and discuss our system DECORUM: a solving laboratory for over-constrained problem.

The paper is organized as follows: section 2 introduces some definitions, section 3 presents our parameterized search method, section 4 details its specialization for CSPs, section 5 discusses the complexity of the approach and section 6 describes DECORUM.

2 Definitions

Let us consider a domain D upon which constraints are defined and a solver S . Let P a property maintained by S throughout the computation. For example, for Constraint Satisfaction Problems (CSP) arc-consistency is the usually maintained property. At each step of the computation (adding a constraint, ...), the property P is enforced.

An **over-constrained** problem is a constraint problem whose constraint system is contradictory (there exists no solution that satisfies all the constraints).

When solving over-constrained problems, constraints may become inactive (they are relaxed) while the others remain active, in order to find a subset of constraints satisfying the user. A **configuration** $\langle A, R \rangle$ is a split of the constraints of a problem in two sets: active constraints (A) and relaxed ones (R). A configuration is said **P -satisfiable** if the property P holds for the set of active constraints. Otherwise, a configuration is said **P -contradictory**.

The user must express its requirements regarding constraints of the problem. He introduces an **hierarchy** [Wilson and Borning, 1993] upon the constraints. This hierarchy is defined using **preferences** between constraints. It is then possible to define **comparators** (partial order relation between configurations) modeling the user's wishes regarding the relaxation of constraints.

A **C -solution**¹ for an over-constrained problem is a P -satisfiable configuration $\langle A, R \rangle$ which maximizes the user-defined comparator C .

3 Best first exploration for over-constrained problems

In this section, we show how a C -solution can be obtained using a search method over the configurations space. This search method ensures that as soon as a P -satisfiable configuration is identified, this configuration is the best one regarding the user-defined comparator C .

The description that follows is an idealized version of the approach. Implementation issues are addressed section 4.

3.1 The configurations space

We consider the configuration space as a binary tree. Each node is a constraint whose children edges represent the two possible states: active or relaxed. This tree is built considering the order of integration of the constraints in the problem (dynamic problems are addressed here).

For a node corresponding to a constraint c_a , an out-coming edge is labeled a (*resp.* a') for the active state (*resp.* relaxed state). Figure 1 shows such a tree for three constraints c_a, c_b and c_c .

Each leaf (or path from the root to a leaf) of this tree represents a *configuration*. For example, the bold path ($a'bc$) from the root to a leaf in figure 1 represents the configuration: $\langle \{c_b, c_c\}, \{c_a\} \rangle$.

Solving an over-constrained problem requires to find the best P -satisfiable configuration regarding the user-defined comparator.

¹Note that when handling CSPs using arc-consistency as property P , enumeration is considered as a set of disjunctive constraints (each variable can have at most one value at a time). The equality constraints (assigning a value to a variable) are dynamically added or retracted. So, the final P -satisfiable configuration $\langle A, R \rangle$ will be such that A contains one equality constraint for each variable. Therefore, such a P -satisfiable configuration will be *satisfiable*.

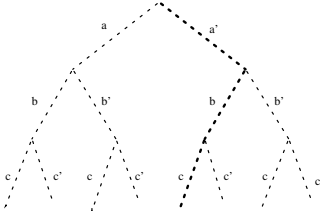


Figure 1: Search space – Binary tree

3.2 Classical Exploration

A naive approach based upon standard backtracking would require an optimality proof step in order to obtain the best configuration.

Intelligent Backtracking (see [Fages *et al.*, 1995; Menezes and Barahona, 1996]) enables the identification of good backtrack points, thus allowing a reduction of the exploration of the search tree. Note that the optimality proof step is always needed.

The main problem introduced by these techniques is **trashing**: already explored and rejected sub-trees are systematically tested upon backtracking. We propose here a search method that uses more information gathered throughout previous computations. We claim that this approach reduces trashing and make an active use of previous computations. We also claim that such an approach can be implemented preserving good complexity results (see section 5).

3.3 A best first approach

In this section, we present our general best first search method. This method is parameterized with the domain on which constraints are defined and the associated solver. It depends on the definition of two functions whose actual implementation will be detailed in section 4.

3.3.1 Contradiction Explanations

Our method relies upon the key concept of **contradiction explanation**. A *contradiction explanation* is a set of constraints whose conjunction leads to a contradiction. It is an *explanation* of the contradiction. This notion of *contradiction explanation* is bound to the property P maintained throughout the computation for the current set of active constraints.

Definition 1 (Contradiction explanation) Let $\langle A, R \rangle$ be the current configuration. We suppose that this configuration has been identified as P -contradictory i.e. $P(A)$ does not hold. Let $E \subset A$ a set of constraints.

E is a contradiction explanation iff $P(E)$ does not hold.

Contradiction explanations are learnt from the identification of a contradiction.

3.3.2 Promising configurations

Let \mathcal{E}_c be the set of all the *contradiction explanations* determined so far during the computation.

Definition 2 (Promising configuration) Let $C_f = \langle A, R \rangle$ a configuration. C_f is a *promising configuration* iff $\forall E \in \mathcal{E}_c, R \cap E \neq \emptyset$. In other words, R covers the set of contradiction explanations.

Proposition 1 (Characterizing a C-solution) *A C-solution is a promising configuration.*

Proof: Let $C_f = \langle A, R \rangle$ a C-solution. Suppose that C_f is not a promising configuration. There exists $E \in \mathcal{E}_c$ such that $E \subset A$. Therefore, C_f would be P-contradictory (A contains an identified *contradiction explanation*) which is not possible. \square

Theorem 1 (Focusing on promising configuration) *Any promising configuration $\langle A, R \rangle$ that maximizes the user-defined comparator C and that is P-satisfiable, is a C-solution for the considered problem.*

Proof: This is a direct consequence of Proposition 1 and the definition of a C-solution. \square

3.3.3 A best first exploration

Now, we assume that there exist two functions: **contradiction-explanation** that extracts a *contradiction explanation* from a P-contradictory configuration and **best-promising-configuration** that computes the² best promising configuration, at the current step of computation, considering the user-defined comparator. We will show in section 4 a comparator and its associated **best-promising-configuration** function.

Let C_f be the current configuration. Let \mathcal{E}_c be the set of computed *contradiction explanations* so far (from the beginning of the dynamic process).

Algorithm 1 (Best-first Exploration)

```

While the current configuration  $C_f$  is P-contradictory do:
  (a) Add contradiction – explanation( $C_f$ ) to  $\mathcal{E}_c$ 
  (b) Let  $C_f \leftarrow$  best – promising – configuration( $\mathcal{E}_c$ )
//  $C_f$  is the best P-satisfiable configuration regarding the user-defined comparator. //

```

This algorithm implements an exploration that can be characterized as a best first search. This search does not need any classical backtrack, only configuration changes are performed. Branches leading to a contradiction are implicitly cut thanks to the *contradiction explanations* base. A lot of unuseful work (configurations) is avoided.

This algorithm terminates because:

- Only new *contradiction explanations* are produced (every tested configuration is a promising one before the identification of the contradiction). So, the exploration will be done only for not yet explored configurations.
- The number of existing configurations is finite: 2^e where e is the number of constraints.
- There exists at least one P-satisfiable configuration: all the constraints are relaxed.

The proposed algorithm is correct because of theorem 1. Note that the enumeration algorithm *Dynamic Backtracking* [Ginsberg, 1993] can be considered as a specialization of our approach limited to a particular kind of constraints: assignment constraints introduced during the enumeration process.

²Note that because a comparator is a partial order relation, there may exist several *best* configurations. Therefore, we choose among these not comparable configurations.

4 Constraint Relaxation for CSPs

Let us recall that a Constraint Satisfaction Problem can be defined as a set \mathcal{V} of variables; the set \mathcal{D} of their respective domains (possible values for each variable) and the set \mathcal{C} of constraints specifying acceptable combinations of values for the variables of the problem.

To specialize our search method for CSP, we need to use a Deduction Maintenance System: a tool that will facilitate the supply of *contradiction explanations*.

4.1 A Deduction Maintenance System

A **deduction** is associated with any of the three following actions:

- Removing a value from the domain of a variable. The deduction associated with the removal of the value a from the domain d_x of the variable x is denoted: $\delta_{(x \neq a)}$.
- Removing a constraint from the constraint store (relaxing). The deduction associated with the relaxation of the constraint c is denoted: $\delta_{(\bar{c})}$.
- Raising a contradiction (the domain of a variable becomes empty). The associated deduction is denoted $\delta_{(\perp)}$.

Let Δ be the set of deductions performed during the resolution.

A **deduction explanation** E , for a deduction δ , is a set of constraints whose conjunction leads to perform the action associated with the deduction.

Definition 3 (Deduction explanation) *Let E be a set of constraints.*

- E is a **deduction explanation** for a deduction $\delta_{(x \neq a)}$ iff a is removed from d_x when achieving P for the set of constraints E .
- E is a **deduction explanation** for a deduction $\delta_{(\perp)}$ iff configuration $\langle E, \emptyset \rangle$ is P -contradictory.
- E is a **deduction explanation** for a deduction $\delta_{(\bar{c})}$ iff configuration $\langle \{c\} \cup E, \emptyset \rangle$ is P -contradictory.

A *deduction explanation* E is **valid** in a given configuration $\langle A, R \rangle$ if $E \subset A$.

A configuration becomes P -contradictory as soon as the domain of a variable becomes empty, *i.e.* all the deduction explanations of the removal of each value in the domain are valid.

4.2 Providing deduction explanations

The simplest *deduction explanation* for any deduction is the complete set of the active constraints of the current configuration. This kind of *deduction explanation* is obviously of no use and would lead to a complete enumeration process of all the possible configurations (that we try to avoid).

The *best deduction explanation* is the minimal set of constraints that verifies the definition 3. When using a particular algorithm during property P enforcement a *good deduction explanation* must reflect the knowledge used by this algorithm to perform any deduction. We detail possible *deduction explanation* when achieving arc-consistency.

Using the AC4 algorithm: The AC4 algorithm [Mohr and Henderson, 1986] uses two main steps to remove unsupported values from the domain of the variables. The first step performs value removals directly due to a unique constraint (thus, the *deduction explanation* is this constraint). In the second step, indirect removals are performed, they are the consequence of the removals in the first step (the *deduction explanation* is then the applied constraint associated with the *deduction explanation* of the propagated removal).

Using the AC5 algorithm: The AC5 algorithm [Van Hentenryck *et al.*, 1992] actively uses the semantics of the handled constraints. This leads to a better *deduction explanation* system. For example, when filtering a constraint $x > y$ considering that the current domain of x is $\{\dots, a\}$, values greater than a will be removed from d_y and the *deduction explanation* for these removals is $E = \bigcup_{b \in d_x, b > a} E_{\delta_{x \neq b}}$.

4.3 Producing Contradiction Explanations

When a contradiction (deduction $\delta_{(\perp)}$) occurs the domain of at least one variable becomes empty. Let x be such a variable. Let note E_δ the *deduction explanation* associated with the deduction δ . Thus $E_{\delta_{(\perp)}} = \bigcup_{a \in d_x} E_{\delta_{(x \neq a)}}$.

Theorem 2 (New contradiction explanation) *When computing a new contradiction explanation for a (previously promising) P-contradictory configuration, only a new contradiction explanation is produced i.e. $E_{\delta_{(\perp)}} \not\subseteq \mathcal{E}_c$.*

Proof: As $E_{\delta_{(\perp)}}$ is valid, it cannot be an already computed *contradiction explanation* because the current configuration would not have been a promising one (not covering the current set of *contradiction explanations*). \square

4.4 Computing a new best promising configuration

As depicted in definition 2, the new best promising configuration must cover the set \mathcal{E}_c of *contradiction explanations* optimizing the comparator-based partial order.

This problem can be modeled as the determination of a set covering in the hypergraph \mathcal{H} :

Definition 4 (The Hypergraph \mathcal{H}) *The hypergraph \mathcal{H} is defined from the set \mathcal{E}_c of contradiction explanations. Each constraint appearing in any element of \mathcal{E}_c is a vertex in \mathcal{H} . Each contradiction explanation E is an hyper-edge in \mathcal{H} .*

The general set covering problem in an hypergraph is NP-complete [Gondran and Minoux, 1990]. In our implementation, we use a *comparator* that simplifies the set covering problem.

Definition 5 (An applicable comparator \prec_c) *Let $C_1 = \langle A_1, R_1 \rangle$ and $C_2 = \langle A_2, R_2 \rangle$ two configurations for a same OCSP.*

$$\begin{aligned} C_1 \prec_c C_2 &\equiv \exists k > 0, \text{ such that} \\ &\forall i < k, R_{1[i]} = R_{2[i]} \\ &R_{1[k]} = \emptyset \wedge R_{2[k]} \neq \emptyset \end{aligned}$$

where $R_{[\ell]}$ is the restriction of R to the constraints with a preference level ℓ in the hierarchy.

The best promising configuration $\langle A', R' \rangle$ regarding \prec_c from a P-contradictory configuration is:

$$\begin{aligned} R' &= \{c \in A \cup R \mid \exists E \in \mathcal{E}_c, c = \min_{\text{pref}}\{c_i \in E\}\} \\ \text{and } A' &= (A \cup R) \setminus R' \end{aligned}$$

Note that using this comparator, best promising configurations can be computed in an incremental way by simply adding a constraint to relax (the least important of the new *contradiction explanation*).

4.5 Performing the configuration jump

When changing the configuration, we compute the set C_α of new active constraints and the set C_β of the new relaxed constraint.

Relaxing a constraint The relaxation of any element of C_β needs to be propagated. The aim of this propagation is to delete the past effects of the relaxed constraints. We use here an extension of specialized algorithms such as DNAC4 (an algorithm that performs constraint suppression in a dynamic environment) [Bessière, 1991].

Let c be an element of C_β . Let Q be a queue. Let $\alpha(c)$ be the subset of Δ (the deductions of the problem) whose validity relies upon c . $\alpha(c) = \{\delta \in \Delta \mid c \in E_\delta\}$

Algorithm 2 (Constraint Relaxation)

- (a) *The deduction explanation for the removal of c is the invalidated contradiction explanation in \mathcal{E}_c (except c)*
- (b) *For each element δ in $\alpha(c)$ do:*
 - (b-1) *If δ is the removal of value a from the domain of the variable x , put back a in d_x and enqueue (x, a) in Q .*
 - (b-2) *If δ is the removal of constraint c enqueue c in Q .*
- (c) *Dequeue element e from Q until Q is empty and do:*
 - (c-1) *If $e = (x, a)$ then test if there exists another way to prove this removal. If so remove a from d_x using the constraint solver.*
 - (c-2) *If $e = c'$ add c' to C_α .*

Completing the jump In order to terminate the jump, every constraint in C_α needs to be introduced in the constraint system. Note that when using our comparator (\prec_c) the constraints in C_α do not need any introduction: a relaxed constraint will never be reintroduced.

4.6 Enumeration and Relaxation

Enumeration in a CSP or an OCSP can be modeled as the dynamic addition/removal of equality constraints (eg. $x = a$ for $a \in d_x$). This approach is completely transparent in our system. After each addition of such a constraint, property P is enforced and if the current configuration is P -contradictory then the constraint relaxation process is initiated handling enumeration constraints in the same way as the other ones. The preference associated to such constraints is the lowest possible so that in case of choice enumeration constraints will be relaxed in priority.

In fact, our algorithm is extended [Jussien and Boizumault, 1996] in order to enforce the introduction of another constraint (testing another value) when relaxing an equality constraint due to the enumeration. This leads to modeling a disjunction between equality constraints. The *deduction explanation* for constraint removal is used in order to ensure the completeness of the approach.

5 Complexity Issues

Let n be the number of variables, e the number of constraints and d the maximum size of the domains. There are $c = e + n \times d$ possible constraints in the problem (we add the equality constraints produced by the enumeration process). There are $a = n \times d + e + 1$ possible deductions in the system.

5.1 Space Complexity

A *deduction explanation* can contain at most c constraints. This is an almost impossible worst case when using *deduction explanations* as defined in section 4.2.

There exists only one *deduction explanation* at a time for deductions associated with value and constraint removals. *Deduction Explanations* for the contradiction deduction represent the set \mathcal{E}_c of *contradiction explanations*. The number of *contradiction explanations* is bounded if the system does not need to reintroduce constraints³. In the worst case, there are c *contradiction explanations*: all the constraints are relaxed one at a time. Thus, there are, in the worst case, $a + c$ *deduction explanations*.

This leads to the following worst case complexity⁴:

$$O(c \times (a + c)) = O(e^2 \times n^2 \times d^2)$$

5.2 Time Complexity

We detail here the time complexity of our approach for the two main parts of our system:

Constraint solving: As we pointed out in section 4.2, the constraint solver (either based upon AC4 or AC5) is modified in order to provide *deduction explanations*. This modification leads to the following worst case time complexity for the modified version of the algorithms.

- AC4: $O(e \times d^3)$ – The original complexity. The *deduction explanation* system used is too simple to decrease the efficiency of the solver, but it enlarges the number of tested configurations before encountering the best.
- AC5: $O(e \times d^2)$ – The time efficiency is decreased because of the *deduction explanation* system, but the search for the best configuration is quicker.

AC5 would be preferred not only because of its best worst case complexity but also because of its active use of the semantics of the treated constraints which provides *better deduction explanations*. A good *deduction explanation* system provides good *contradiction explanations* (*i.e.* not too general ones). The number of explored configurations is then decreased.

Configuration jump: This jump is performed in two steps: an identification step that produces the new *contradiction explanation* and determines the target configuration and a re-computation step that performs the configuration change.

The production of the new *contradiction explanation* is immediate. The determination of the new configuration can be performed in $O(c)$ (the least important constraint in the new *contradiction explanation*) when using the \prec_c comparator.

The configuration change step depends on the cardinal of the sets C_α and C_β as defined earlier in the paper. When using \prec_c , C_α is empty and C_β is a singleton.

³This depends on the comparator. \prec_c is a good one.

⁴The worst case complexity is the same whatever the constraint solver is (AC4 or AC5).

The worst case time complexity of the constraint relaxation propagation is the same as for DNAC4: $O(\epsilon \times d^2)$.

The resulting worst case time complexity for the configuration jump is:

$$O(c + \epsilon \times d^2) = O(\epsilon \times n \times d^2)$$

Although complexities are similar for different solvers, the number of tested configurations strongly depends on it. The more *intelligent* the solver is, the lower the number of tested configurations will be.

6 The DECORUM system

The DECORUM (Deduction-based Constraint Relaxation Management) system is an implementation of our search method for dynamic CSPs. Two versions have been implemented, one using MIT Scheme and the other using Yves Caseau's language CLAIRE [DMI, 1996].

We used DECORUM to experiment and validate our ideas. Our system can easily handle problems involving hundreds of variables and thousands of constraints because the practical complexities of our approach are far below the worst-case values we presented here.

7 Related Works

7.1 Constraint Logic Programming

Hierarchical Constraint Logic Programming (HCLP) [Wilson and Borning, 1993] has a central role in the CLP community. HCLP is an extension of CLP where each constraint is provided with a level of preference thus defining a constraint hierarchy. The terminology of HCLP is widely used by approaches dealing with hierarchical systems.

The operational behavior of HCLP is like the building of a *tower of constraints*. Constraints are introduced level by level in the system (from the most important to the less important ones) until an inconsistency appears. This approach needs to completely know the constraint system before resolution (this is not suitable for dynamic environments). Moreover, operational HCLP as proposed by Wilson and Borning, requires a complete solver, otherwise every addition of constraint leads to a complete enumeration in order to prove the solvability.

Menezes and Barahona [1996] proposed an instance of HCLP over Finite Domains called Incremental Hierarchical Constraint Solver (IHCS). This approach explores a set of *promising* configurations but these do not fulfill the requirements of our definition. IHCS leads to more configuration tests than our method. Moreover, such an approach does not avoid trashing because new configurations are tested for satisfiability from scratch.

7.2 Constraint Satisfaction Problems

In the field of CSP, Partial Constraint Satisfaction Problems (PCSP) [Freuder, 1989] are a leading research topic. In the PCSP approach *metrics* are defined in order to optimize the selection of the smallest set of constraints to be relaxed in order to find a solution. Main PCSP works deal with a metric called Maximal Satisfiability which selects solutions which maximize the total number of satisfied constraints without trying to distinguish between ones which are more or less important. This approach also need to completely know the constraint system before any resolution because it uses branch and bound techniques.

Other formalisms (not suitable for dynamic problems) have been proposed, including Schiex's Possibilistic CSP [1992] and Fargier's Fuzzy CSP [Fargier *et al.*, 1993].

8 Conclusion and Further Works

We proposed in this paper a best first search method over configurations in order to handle constraint relaxation in dynamic environments. Moreover, this strategy can be parameterized by a domain and a constraint solver. We show through the DECORUM system that our approach could be used in practice for over-constrained CSP. Complexity results are described.

Now, we plan to work on the following:

- An operational semantics of our transformations in order to embed our approach in Logic Programming,
- The use of our system to efficiently handle disjunctions as it is done for the domain of a variable in DECORUM,
- The extension of our approach to handle constraints on other domains (such as Intervals).

References

- [Bessière, 1991] Christian Bessière. Arc consistency in dynamic constraint satisfaction problems. In *Proceedings AAAI'91*, 1991.
- [DMI, 1996] DMI-ENS – Département Mathématiques et Informatique – École Normale Supérieure, Paris, France. *Introduction to the CLAIRE programming language*, 1996.
- [Fages *et al.*, 1995] François Fages, Julian Fowler, and Thierry Sola. A reactive constraint logic programming scheme. In *International Conference of Logic Programming, ICLP'95*, Tokyo, 1995.
- [Fargier *et al.*, 1993] Hélène Fargier, Jérôme Lang, and Thomas Schiex. Selecting preferred solutions in Fuzzy Constraint Satisfaction Problems. In *EUFIT'93: Proceedings 1st European Congress on Fuzzy and Intelligent Technologies*, Germany, 1993.
- [Freuder, 1989] Eugene Freuder. Partial constraint satisfaction. In *IJCAI-89: Proceedings 11th International Joint Conference on Artificial Intelligence*, pages 278–283, Detroit, 1989.
- [Ginsberg, 1993] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.
- [Gondran and Minoux, 1990] Michel Gondran and Michel Minoux. *Graphes et Algorithmes*. Direction des Études et Recherche d'Électricité de France, Éditions Eyrolles, 1990.
- [Jussien and Boizumault, 1996] Narendra Jussien and Patrice Boizumault. Implementing constraint relaxation over finite domains using ATMS. In Michael Jampel, Eugene Freuder, and Michael Maher, editors, *Over-Constrained Systems*, number 1106 in Lecture Notes in Computer Science, pages 265–280. Springer-Verlag, 1996.

- [Menezes and Barahona, 1996] Francisco Menezes and Pedro Barahona. Defeasible constraint solving. In Michael Jampel, Eugene Freuder, and Michael Maher, editors, *Over-Constrained Systems*, number 1106 in Lecture Notes in Computer Science, pages 151–170. Springer Verlag, 1996.
- [Mohr and Henderson, 1986] R. Mohr and T. C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.
- [Schiex, 1992] Thomas Schiex. Possibilistic constraint satisfaction problems or “How to handle soft constraints?”. In *8th International Conference on Uncertainty in Artificial Intelligence*, Stanford, July 1992.
- [Van Hentenryck *et al.*, 1992] 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.
- [Wilson and Borning, 1993] Molly Wilson and Alan Borning. Hierarchical constraint logic programming. *Journal of Logic Programming*, 16(3):277–318, July 1993.