# Constraint solving in uncertain and dynamic environments - a survey

Gérard Verfaillie, Narendra Jussien

# Constraint Solving in Uncertain and Dynamic Environments: A Survey

Gérard Verfaillie (gerard.verfaillie@laas.fr)
*Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS), Toulouse, France*

Narendra Jussien (narendra.jussien@emn.fr)
*École des Mines de Nantes (EMN), Nantes, France*

February 12, 2005

**Abstract.**

This article follows a tutorial, given by the authors on *dynamic constraint solving* at CP 2003 (Ninth International Conference on Principles and Practice of Constraint Programming) in Kinsale, Ireland (Verfaillie and Jussien, 2003).

It aims at offering an overview of the main approaches and techniques that have been proposed in the domain of *constraint satisfaction* to deal with *uncertain* and *dynamic* environments.

**Keywords:** Constraint Satisfaction Problem, Uncertainty, Change, Stability, Robustness, Flexibility

## 1. Why dynamic constraint solving?

### 1.1. A first global view

The *constraint satisfaction problem* (CSP) (Mackworth, 1992; Dechter, 1992) framework has been proposed as a generic way of modelling discrete constrained decision problems, for which generic deduction and search algorithms can be defined. The framework, as well as the associated algorithms, assume that all the components of the instance to consider (variables, domains of possible values, constraints to satisfy) are completely known before modelling and solving it and do not change either during or after modelling and solving. However, it has been observed for a long time that such assumptions do not hold in many situations, specifically each time one has to deal with uncertain and dynamic environments. See for example (Montanari and Rossi, 1997).

### 1.2. The example of online planning

Because *online planning and scheduling* is a rich context where uncertainties and changes cannot be easily avoided, we propose to use it as

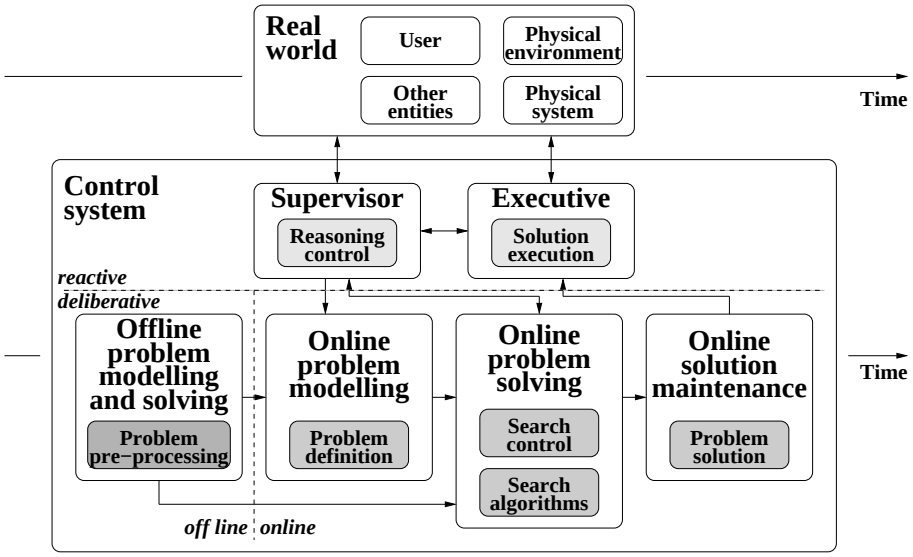


*Figure 1.* Architecture of the high-level control system of a physical system.

an introductory example. Figure 1 shows the classical architecture of the high-level control system of either an autonomous artefact (robot, aircraft, satellite, etc.) or a large system (production, distribution, traffic, etc.). See for example (Alami et al., 1998; Muscettola et al., 1998b). We can observe two main blocks:

1. the *real world*, which includes the controlled *physical system* itself, its *environment*, as well as its *users* and other *related entities* which may be either human or artificial;

2. the high-level *control system*.

Observe that these blocks are both plunged into time and can be seen as two interconnected *concurrent processes*. The term *online*, widely used in the *automatic control* community, refers to such a situation: a control system running concurrently with the real world. The control system is itself composed of three main parts:

1. a *reactive* part, composed of a *supervisor* and an *executive*. Modules in this part are designed to react within a strictly limited time to events coming from either the real world or the control system itself. One usually speaks of (*hard*) *real-time* modules;

2. an *online deliberative* part, composed of three modules: *problem definition*, *problem solving*, and *solution maintenance*. Even if tem-

poral constraints may limit the time taken by the reasoning activities of these modules, these constraints are generally softer than the ones of the reactive part. One often speaks of *soft real-time* modules;

3. an *offline deliberative* part, only composed of a *problem modelling and solving* module. A priori, no temporal constraints or only very soft ones limit the time taken by the reasoning activities of this module.

If we consider now these modules one after the other, we can note that:

- the *supervisor* can be seen, as suggested by its name, as the core of the control system, in charge of activating and controlling its reasoning;
- the *online problem definition* module is in charge of modifying when needed the definition of the problem(s) to solve, as a function of the information coming from the real world via the supervisor;
- the *online problem solving* module is in charge of solving the problem(s) defined by the previous module, whenever the supervisor requires it; to be able to meet the requirements of the supervisor, it is composed of a *search control* submodule which controls in turn a set of parameterised deduction and search algorithms;
- the *online solution maintenance* module is in charge of the maintenance of the solution(s) produced by the previous module;
- the *executive* can be seen as the interface between the control system and the physical one, in charge of the execution of the solution(s) produced and memorised by the previous modules;
- finally, the *offline problem modelling and solving* module can be seen as a form of problem preprocessing, which may simplify and enhance online problem solving.

One of the difficulties of problem modelling and solving in such a context comes from the fact that, on the one hand, the knowledge about the real world is often incomplete, imprecise and uncertain and, on the other hand, the real world and the knowledge about it may change during or after modelling and solving.

To be more concrete, consider a *travel management system* (TMS), embedded in a car, in charge of the management of all the features of

a long travel: route, stops, reservations, rendezvous, car refuelling and maintenance, etc. The physical system is the car. The user is the car driver. Its physical environment is the road, the traffic, the weather, etc. Other entities are hotels, restaurants, garages, other people and similar TMSs. Uncertainty may come from the car (actual state of its components), from the environment (actual state of the road, future traffic and weather), and from the other entities (actual availability). Changes may occur at any time from the driver herself (changes in her goals or in her current plan), from the car (unexpected breakdowns), from the environment (traffic jams), and from the other entities (unavailability, rendezvous cancellations).

### 1.3. Various contexts

Handling uncertainties and changes is necessary not only in online planning and scheduling, but also in many other contexts. Some of them appear in real-world applications, such as:

- *online computer vision, failure diagnosis, and situation tracking*, with uncertainties about the actual state of what is observed (for example, the actual state of the components of a physical system) and with changes that may occur at any time in the current set of observations (for example, occurrence of a failure symptom);
- *computer-aided system design or configuration*, with uncertainties about the environment in which the designed system will be used, about the actual properties of the chosen components, and about the actual physical constraints to meet, and with changes that may occur either in user requirements or in designer choices (Lottaz et al., 1999; O'Sullivan, 2002);
- *user interface management*, with changes that may occur at any time due to actions on the interface by the user or the software (Borning and Freeman-Benson, 1998);
- *interactive* or *distributed problem solving*, with uncertainties about the decisions of the user (in interactive applications) or of the other entities (in distributed applications) and with changes that may result from such decisions (Yokoo et al., 1998).

Furthermore, other ones appear in the process of problem modelling or solving, independently of any application domain, such as:

- *interactive problem specification* and *constraint program debugging*, with changes that may result from the problem specification or modelling that the user chooses (Deransart et al., 2000);

– any form of *unordered local search*, such as *dynamic backtracking* (Ginsberg, 1993) or *decision repair* (Jussien and Lhomme, 2002), with changes that come from variable assignments or unassignments.

## 2. Survey contents

In Section 3, we list the main requirements that appear when handling uncertainties and changes in constraint solving. In Section 4, we present the frameworks that have been proposed to represent dynamic and uncertain constraint satisfaction problems. Then, in Sections 5 and 6, we describe the two classes of methods that have been proposed to deal with uncertainties and changes: reactive methods which aim at reusing solutions or reasoning, and proactive ones which aim at producing robust or flexible solutions. In Section 7, we underline interesting research directions. Finally, in Section 8, we consider connections with studies on related frameworks.

Note that studies have been carried out for many years in the planning and scheduling domain on the ways of handling correctly and efficiently uncertainties and changes. Without claiming exhaustivity, we will refer to some of them.

## 3. Main associated requirements

In uncertain and dynamic situations, four kinds of requirements appear, even if only some of them may be present in a specific application:

1. the first one is *to limit as much as possible the need for successive online problem solvings*, because online problem solving may be consuming in terms of computing time and resources and disturbing in terms of solutions: changes in produced solutions may become too frequent. For example, in the TMS application, the driver does not want her route or her schedule be too frequently changed, because of successive small changes in the travelling conditions;

2. the second one is *to limit as much as possible changes in the produced solutions* when the first approach failed, i.e. when the previous solution is no more valid and producing online a new one is necessary. This is because changes that are too important are undesirable. In the TMS application, the driver does not want her route and her schedule be completely changed, because of a small change in the travelling conditions;

3. the third one is *to limit as much as possible the computing time and resources* that are necessary for online problem solving, again when the first approach failed. This is because, in many applications, the utility of a solution decreases with the time of its delivery. For example, in online planning and scheduling applications, the time taken for reasoning may be lost for acting. Moreover, in some applications, the utility of a solution may become null after a given time. In the TMS application, telling the driver to leave a highway at a given exit is useless if she gets the information after having passed by this exit;

4. the fourth one is *to keep producing consistent and optimal solutions*. Consistency is necessary when physical constraints and hard user requirements are concerned. Optimality is desirable even in an online problem solving context. Always, in the TMS application, the driver does not wish to see her route and her schedule completely degraded after a small change in the travel conditions. She wishes also that better conditions be exploited to produce better solutions. Note that this last requirement may interfere with the second one: possible contradiction between optimality and stability of a solution.

## 4. Modelling frameworks

The notion of *dynamic constraint satisfaction* has been first introduced in (Dechter and Dechter, 1988). A *dynamic constraint satisfaction problem* (DCSP) is a sequence of CSPs, each one resulting from some changes in the definition of the previous one. These changes may affect any component in the problem definition: variables (addings or removals), domains (changes in the intensional definition, value addings or removals in case of extensional definition), constraints (addings or removals), constraint scopes (variable addings or removals), or constraint definitions (changes in the intensional definition, tuple addings or removals in case of extensional definition). Because domains can be seen as unary constraints, because variables are implicitly added or removed with all the constraints that apply to them, and because any change in a component can be seen as a removal followed by an adding, all these changes can be basically expressed in terms of constraint addings or removals. More formally, a DCSP is thus a sequence $\{P_0, P_1, \ldots, P_i, \ldots, P_n\}$ where, for each $i$, $1 \leq i \leq n$, $P_i$ is a CSP, $Ca_i$ is a set of added constraints, $Cr_i$ is a set of removed constraints, such that $Cr_i \subseteq P_{i-1}$, and $P_i = P_{i-1} + Ca_i - Cr_i$. Note immediately that

this definition is extremely general. Because nothing prevents $Cr_i$ from being the set of constraints in $P_{i-1}$, it covers any sequence of CSPs. In fact, it is implicitly assumed that changes are limited with regard to the problem definition: only a small fraction of the constraints are added or removed at each step $i$.

This framework must not be mistaken for extensions of the basic CSP framework that have been proposed to capture more easily static real-world problems, such as:

- the *dynamic constraint satisfaction problem* framework proposed in (Mittal and Falkenhainer, 1990), renamed *conditional constraint satisfaction problem* (CCSP) in (Sabin and Freuder, 1998), and studied in (Bowen and Bahler, 1991; Sabin and Freuder, 1996; Sabin and Freuder, 1999; Soininen et al., 1999; Gelle and Faltings, 2003; Sabin et al., 2003). The basic objective of the CCSP framework is to model problems whose solutions do not all have the same structure, i.e. do not all involve the same set of variables and constraints. Such a situation occurs when dealing with product configuration or design problems, because the physical systems that can meet a set of user requirements do not all involve the same components. More generally, it occurs when dealing with any synthesis problem, such as design, configuration, planning, scheduling, etc. In a CCSP, the set of variables is divided into a set of mandatory variables and a set of optional ones. The set of constraints is also divided into a set of compatibility constraints and a set of activity constraints. Compatibility constraints are classical constraints. Activity constraints define the conditions of activation of the optional variables as a function of the current assignment of other mandatory or optional variables. Constraints are activated only if their variables are activated too. When solving a CCSP, the structure of the problem (activated variables and constraints) may change as a function of the current assignment. Thus, a CCSP can be considered as a particular case of DCSP where all the possible changes are defined by the activity constraints. However, the methods that have been proposed so far for dealing with DCSP and with CCSP are very different from each other;

- the *open constraint satisfaction problem* framework (OCSP) proposed in (Faltings and Macho-Gonzalez, 2002), named *interactive constraint satisfaction problem* in (Lamma et al., 1999). In an OCSP, the allowed values in domains, as well as the allowed tuples in relations, may not be all known when starting a search for a solution. They may be acquired online when no solution has been found with the currently known values and tuples. Such a situation

occurs each time the acquisition of information about domains and relations is a costly process that needs either heavy computation or requests to distant sites. Thus, an OCSP can be considered as a particular case of DCSP where all the possible changes result in extension of the domains and relations.

The DCSP framework also differs from frameworks that aim at including in the problem definition the available knowledge about possible changes from the real world, such as:

- the *mixed constraint satisfaction problem* framework (MCSP) proposed in (Fargier et al., 1996) to model decision problems under uncertainty about the actual state of the real world. In an MCSP, variables are divided into controllable variables (decision variables) that are under the control of the decisional agent and uncontrollable variables (state variables) that are not under its control. In such a framework, a basic request may be to build a decision (an assignment of the decision variables) that is consistent whatever the state of the world (the assignment of the state variables) is;

- the *probabilistic constraint satisfaction problem* framework (PCSP) proposed in (Fargier and Lang, 1993) to model decision problems under uncertainty about the presence of constraints. In a PCSP, a probability of presence in the real world is associated with each constraint. In such a framework, a basic request may be to produce an assignment that maximises its probability of consistency in the real world. A PCSP is a particular case of *valued constraint satisfaction problem* (VCSP) (Schiex et al., 1995; Bistarelli et al., 1999);

- the *stochastic constraint satisfaction problem* framework (SCSP) proposed in (Walsh, 2002) to model decision problems under uncertainty about the actual state of the real world, the same way as the MCSP framework. The SCSP framework is inspired from the *stochastic satisfiability problem* (SSAT) (Littman et al., 2001). As in an MCSP, variables are, in an SCSP, divided into controllable ones (decision variables) and uncontrollable ones (state variables). The main difference from an MCSP is that a probability distribution is associated with the domain of each state variable. Another difference is that requests can freely alternate state and decision variables. In such a framework, a basic request may be, as in the PCSP framework, to build a decision (an assignment of the decision variables) that maximises its probability of consistency in the real world;

– the *branching constraint satisfaction problem* framework (BCSP) proposed in (Fowler and Brown, 2000) to model sequential decision problems under uncertainty about the arrival of new elements (objects, tasks ...). In a BCSP, at each step, present variables are assigned when possible, taking into account variables that will be added to the problem definition. A utility is associated with each assigned variable. At each step, a probability of addition is associated with each absent variable. The request is, at each step, to assign the added variable a value that maximises the global expected utility. In (Fowler and Brown, 2003), a BCSP is viewed as a particular case of *Markov decision process* (Puterman, 1994).

Note also the difference from problems that result from the modelling of planning, scheduling, diagnosis, or situation tracking problems as CSPs. Although these problems are considered by some authors as dynamic ones (Miguel and Shen, 2003), they can also be considered as static ones because all the possible changes in the real world (future or past) can be modelled in a unique static CSP. A planning problem, without uncertainty about the initial state and the action effects, can indeed be modelled as a static CSP. See for example (van Beek and Chen, 1999). In case of uncertainty, it can be modelled as a MCSP or SCSP if knowledge about the possible changes is available at the planning time, or as a DCSP otherwise.

We already observed that the DCSP framework is very general, in that it assumes nothing about the changes that may occur. No information must be provided by the modeller about their nature or their absolute or relative likelihood of occurrence. Changes may be completely unexpected. For example, in the TMS application, this is the case of a serious breakdown on a new car, which is usually not taken into account in the planning process, because it is too rare to influence the decision, unless a travel across a wide desert is planned. This framework is the basis of the so-called *reactive* approaches that are presented in Section 5.

On the other hand, frameworks such as MCSP, PCSP, SCSP, or BCSP are more restrictive, but richer, in that they assume that the modeller provides the system with information about the possible changes and sometimes about their absolute or relative likelihood of occurrence. For example, in the TMS application, this is the case with future weather and traffic for which forecast or statistical models may be available via connection to distant information sites and may influence the decision. These frameworks are the basis of the so-called *proactive* approaches that are presented in Section 6.

## 5. Reactive approaches

### 5.1. Basic principles

Under the term *reactive*, we gather all the methods that use no knowledge of the possible changes. This does not mean that these methods do not try to anticipate the possible changes, for example by recording when solving some potentially useful information. This only means that they use no information about the possible directions of the future changes. The drawback of this absence of knowledge may be a lack of robustness of the produced solutions. The advantage may be, on the contrary, an ability to react to any kind of change.

The basic principle of these methods, when facing a new problem after a change in the problem definition, is to try reusing as much as possible what has been produced by previous problem solvings. Now, if we review what can be produced when solving a problem, we get four kinds of information:

1. local consistency or inconsistency of partial assignments, according to the used level of local consistency;

2. global consistency or inconsistency of partial assignments, i.e. the ability or not to be extended into solutions;

3. consistency or inconsistency of complete assignments, i.e. the fact of being a solution or not;

4. problem consistency or inconsistency, i.e. the global consistency or inconsistency of the empty assignment.

Note the symmetric behaviour of consistency and inconsistency information with regard to problem relaxation and restriction, i.e. constraint removal and adding. Consistency information is preserved by problem relaxation (what was consistent remains consistent), but inconsistency information is not (what was inconsistent may either remain inconsistent or become consistent). Conversely, inconsistency information is preserved by problem restriction, but consistency information is not. In case of simultaneous or sequential problem relaxation and restriction, for example in case of a change in the definition of a constraint invalidating the current solution, everything is lost (consistency and inconsistency information).

Roughly speaking, *solution reuse* techniques (sections 5.2 and 5.3) try reusing consistency information, whereas *reasoning reuse* techniques (sections 5.4 and 5.5) try reusing inconsistency information. Obviously,

hybrid techniques, which try reusing both consistency and inconsistency information, are possible (section 5.6). From previous observations related to problem relaxation and restriction, we can deduce that the challenge of solution reuse techniques is to deal with problem restriction (nothing has to be done in case of relaxation), whereas the one of reasoning reuse techniques is to deal with problem relaxation.

### 5.2. Solution reuse techniques

Solution reuse techniques deal with situations where (i) the previous problem $P$ has been solved and proved to be consistent, (ii) a solution $S$ of $P$ has been produced and recorded, (iii) a change occurred in the definition of $P$, resulting in a new problem $P'$, but, (iv) due to some problem restriction, $S$ is no longer a solution. Note that, if $P'$ is a relaxation of $P$, $S$ is obviously solution of $P$ and $P'$. The basic assumption of solution reuse techniques is that, because $P'$ is close to $P$, a solution $S'$ of $P'$ can be reasonably searched for in the vivinity of $S$. If this assumption is correct, this approach favours both search efficiency and solution stability, i.e. the second and the third of the above requirements (see Section 3). It is however easy to build counterexamples where a small change in the problem definition results in a minimum distance (*Hamming* distance for example, i.e. the number of different variable assignments) between a solution of $P$ and a solution of $P'$ that is as high as required. As confirmed by experiments, such an approach works in most cases, but may become sometimes very inefficient. Although every classification is arguable, we can distinguish in the literature four kinds of techniques of this type:

1. *tree search-based* methods. The most immediate way of reusing a previous solution in the context of depth-first tree search algorithms, is to use it as a heuristics. In the new tree search, for each variable, the value assigned to it in the previous solution is first chosen. This is for example what is done in (Hentenryck and Provost, 1991). It is however well known that a depth-first tree search does not explore the space of possible assignments according to an increasing distance to the first heuristic assignment. So, using a *limited discrepancy search* algorithm (Harvey and Ginsberg, 1995) with an increasing limit is certainly more appropriate in such a context.

2. *local search-based* methods. Local search algorithms are certainly those that fit the best a dynamic context. The previous solution can simply be used as a starting assignment for the new local search. Because local search is basically a repair process that tries to adapt

an assignment to a set of constraints, there is no difference, from the algorithm point of view, between, on the one hand, a usual move that results from a change in the current assignment and, on the other hand, a change in the current set of constraints to consider. The local search version of *min-conflicts* (Minton et al., 1992) is an example of such a method.

3. *solution perturbation-based* methods. Although their application area is potentially large, solution perturbation methods have been first developed in the particular context of user interface management applications (Freeman-Benson et al., 1990). Their principle is to synthesise offline a sequence of methods that can be then straightforwardly applied online in case of any change that invalidates the previous solution, in order to transform it into a solution of the new problem. They are basically limited to acyclic constraint networks that involve only functional one-way constraints, also referred to as *dataflow* constraints: several input variables and one output variable whose value is a function of the values of the former. Several extensions have been however proposed to deal with cyclic networks involving multi-way non functional constraints (Sannella et al., 1993; Borning and Freeman-Benson, 1998; Trombettoni, 1998).

4. *variable unassignment and reassignment-based* methods like *local changes* (Verfaillie and Schiex, 1994b). As do local search and solution perturbation-based methods, these methods start from the previous solution that they try to transform into a solution of the new problem. They differ however in their way of doing that. Their principle is to identify unsatisfied constraints, to unassign at least one variable for each unsatisfied one, and then to reassign the unassigned variables one after the other. A variable reassignment is performed by choosing a value for it, by evaluating the resulting unsatisfied constraints, by unassigning the same way at least one variable for each unsatisfied constraint, and then to reassign the unassigned variables one after the other, by avoiding unassigning the variables that unassigned them (recursive process). In spite of its local search flavour, this method, as *min-conflicts* (Minton et al., 1992), terminates and is complete when embedded in a tree search. Note the proximity between these methods and scheduling methods that use an iterative repair approach to deal with changes (Smith, 1995; Zweden et al., 1994; Chien et al., 2000; Kramer and Smith, 2003), although the latter generally lay down completeness for efficiency.

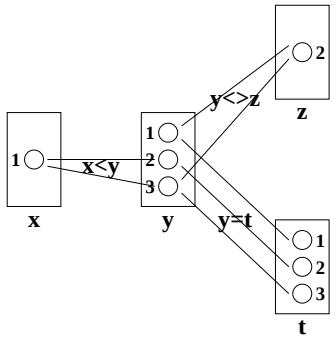


*Figure 2.* A small instance.

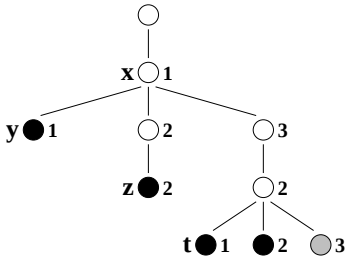


*Figure 3.* Depth-first tree search.

### 5.3. An example of solution reuse

We use the small instance in Figure 2 to illustrate these techniques. This instance involves 4 variables $x$, $y$, $z$, and $t$. The domains of $x$ and $z$ are limited to a singleton: 1 for $x$ and 2 for $z$. On the contrary, the domains of $y$ and $t$ involve 3 values: 1, 2, and 3. Moreover, this instance involves 3 constraints: $x < y$, $y \neq z$, $y = t$. Figure 2 shows the *consistency graph* of this instance, with edges representing allowed pairs of values. Note that no constraints exist between $x$ and $z$, $x$ and $t$, and $z$ and $t$. All the pairs of values are thus allowed, but associated edges are omitted.

We assume that the constraint $x < y$ does not exist at time 1 and that the solution $\{x = 1, y = 1, z = 2, t = 1\}$ has been produced. We assume that this constraint, which is not satisfied by the solution produced at time 1, appears at time 2 (problem restriction).

Figure 3 shows the tree that is developed at time 2 by a depth-first tree search using only backward-checking and the static orders $\{x, y, z, t\}$ on the variables and $\{1, 2, 3\}$ on the values. Each failure node is shown in black and the first leaf success node is shown in grey.

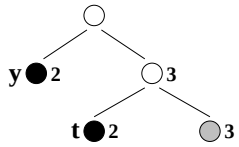


*Figure 4.* Variable unassignment and reassignment.

Figure 4 shows the tree that is developed at the same time 2 by a variable unassignment and reassignment-based method, like *local changes* (Verfaillie and Schiex, 1994b), starting from the solution produced at time 1 and using backward-checking and the same variable and value orderings. At the beginning of the search, only constraint $x < y$ is violated. Because $x$ has only one possible value, it cannot be unassigned and the only thing to do is to unassign $y$ and to assign it another value. This is what is tried at the first level of the tree search. If we choose value 2 for $y$, constraint $x < y$ is no longer violated, but constraints $y \neq z$ and $y = t$ are now violated. Because $y$ can no longer be unassigned in this branch (it has been already unassigned and reassigned), $z$ and $t$ must be unassigned, in order to assign them another value. But, because $z$ has only one possible value, it cannot be unassigned and failure is unavoidable in this branch. A backtrack occurs. If we choose now value 3 for $y$, constraint $x < y$ is no longer violated and only constraint $y = t$ is now violated. Because $y$ can no longer be unassigned in this branch, $t$ must be unassigned, in order to assign it another value. This is what is tried at the second level of the tree search. If we choose value 2 for $t$, constraint $y = t$ is still violated. Because $y$ and $t$ can no longer be unassigned in this branch (they have been already unassigned and reassigned), failure is unavoidable. Another backtrack occurs. But if we choose value 3 for $t$, all the constraints are now satisfied. A solution has been found for the new problem.

Note that similar differences between the trees developed by a usual depth-first tree search and by a variable unassignment and reassignment-based method would appear when using other variable and value orderings, as well as filtering methods such as *forward-checking* or *arc consistency* enforcing.

### 5.4. Reasoning reuse techniques

Reasoning reuse techniques deal with situations where (i) the previous problem $P$ has been solved and proved to be either consistent or inconsistent, (ii) this solving allowed a set $I$ of implied constraints (consequences of the constraints in $P$) to be produced and recorded, (iii)

a change occurred in the definition of $P$, resulting in a new problem $P'$, but, (iv) due to some problem relaxation, the validity of $I$ is no longer guaranteed: for each constraint in $I$, we are not sure that it is implied by the constraints in $P'$. Note that, if $P'$ is a restriction of $P$, $I$ is obviously valid in $P$ and $P'$. The basic assumption of reasoning reuse techniques is that, because $P'$ is close to $P$, most of the constraints in $I$ remain valid in $P'$. If this assumption is correct, keeping recorded the ones that remain certainly valid and avoiding producing them again favour search efficiency, i.e. the third of the above requirements (see Section 3). It is however easy to build counterexamples where a small change in the problem definition invalidates all the constraints in $I$. Experiments confirm that such an approach allows computing to be saved, especially when dealing with constrained problems for which producing $I$ is costly.

All these techniques involve two phases: a first one where the set $I''$ of the constraints in $I$ that do not remain certainly valid is removed and a second one where a new set $I'$ of implied constraints is produced from the constraints in $P'$ and $I \setminus I''$ (the constraints in $I$ that remain certainly valid). Reasoning reuse techniques differ from each other only in the information they use to determine whether an implied constraint in $I$ becomes questionable: constraint graph, constraint justification, or constraint explanation. This results in three families of methods:

1. *graph-based* methods. See for example (Berlandier and Neveu, 1994; Georget et al., 1999). They use only constraint graph information to determine whether an implied constraint becomes questionable. When an initial constraint $c$ is removed or when an implied constraint $c$ is removed because its validity is no longer guaranteed, all the implied constraints that may have been produced by using $c$ according to the constraint graph information become questionable and may be removed too, with a possible propagation in the constraint graph. For example, in the context of arc consistency, when a value is reintroduced in the domain of a variable $v$, all the value removals in the domains of the variables that are connected to $v$ become questionable.

2. *justification-based* methods. See for example (Prosser, 1989; Prosser et al., 1992; Bessière, 1991; Bessière, 1992; Debruyne, 1996; Fages et al., 1998). Based on a TMS-like (*truth maintenance system*) approach (Doyle, 1979), they use the justification that has been recorded with each implied constraint $c'$ to determine whether $c'$ becomes questionable. The justification of an implied constraint $c'$ is simply the constraint $c$ or the set $C$ of constraints that directly produced it. For example, in the context of arc consistency, the

justification of the removal of a value $val'$ from the domain of a variable $v'$ is the constraint $c$ that links $v'$ to another variable $v$, such that no support has been found for $val'$ in the current domain of $v$ when enforcing arc consistency. As with graph-based methods, when an initial constraint $c$ is removed or when an implied constraint $c$ is removed because its validity is no longer guaranteed, all the implied constraints whose justification is $c$ or includes $c$ become questionable and may be removed as well, with a possible propagation in the constraint graph. For example, in the context of arc consistency, when a value is reintroduced in the domain of a variable $v$, any value removal in the domain of any variable $v'$ connected to $v$ by a constraint $c$, whose justification is $c$, becomes questionable.

3. *explanation-based* methods. See for example (Schiex and Verfaillie, 1993; Jussien et al., 2000; Jussien and Lhomme, 2002; Debruyne et al., 2003). Also based on a TMS-like approach, they differ from the previous justification-based methods only in the kind of information that is recorded with each implied constraint $c'$. The recorded information is an explanation of $c'$, i.e. a set $C$ of initial constraints that logically imply it. With this more complete information, when an initial constraint $c$ is removed, all the implied constraints whose justification includes $c$ are removed in one phase, without propagation.

We can observe that justification and explanation-based methods are more time and space consuming when producing implied constraints and their justifications or explanations. However, they result in a more precise information that allows fewer constraints in case of justifications, and even no constraints in case of explanations, to be checked in order to maintain valid implied constraints in case of problem relaxation. We must also note that recorded justifications or explanations strongly depend on the order in which variables and constraints are handled by the implied constraint production algorithms.

A synthesis of some of these methods in the context of arc consistency enforcing for dynamic CSP can be found in (Debruyne, 1996) (short version) or in French in (Debruyne, 1995) (longer version).

### 5.5. An example of reasoning reuse

We use the same small instance in Figure 2 to illustrate these techniques. The only difference with Section 5.3 is that we invert instances at time 1 and 2. We assume that constraint $x < y$ exists at time 1, that

arc consistency filtering has been performed and that this constraint disappears at time 2 (problem relaxation).

Figure 5 (1) shows the result of arc consistency filtering at time 1. Removed values are shown in black. Figure 5 (2a) shows in grey the value removals that become questionable during the first step of arc consistency filtering at time 2, using a graph-based method like $AC|DC$ (Berlandier and Neveu, 1994). Because constraint $x < y$ has been removed, removals of values 1 and 2 from $y$'s domain become questionable. Because value 1 had no support in $x$'s domain according to the removed constraint, it must be restored. But, because value 2 had at least one support in $x$'s domain according to the removed constraint, it can remain removed (it cannot have been removed because of constraint $x < y$). Then, because value 1 has been restored in $y$'s domain, removals of values 1 and 2 from $t$'s domain become questionable too. Because value 1 is supported by the restored value, it must be restored. But, because value 2 is not supported by the restored value, it can remain removed. Figure 5 (2b) shows in black the values that remain removed at the end of this first step. It shows also the result of the second step of arc consistency filtering at time 2, because none of the restored values is removed again during the second step. Note that generally this is not the case: values that are restored during the first step may be removed during the second one. Note also that, by using this graph-based method, all the values that are removed by arc consistency filtering at time 1 become questionable during the first step of arc consistency filtering at time 2. However, generally this is not the case. Note finally that this method needs to determine whether a questionable value is supported by another value, in order to determine whether it must be restored. This may result in costly constraint checks.

Figure 6 (1) shows the result of arc consistency filtering at time 1, using a justification-based method like $DnAC6$ (Debruyne, 1996), which associates with each removed value the constraint that justifies this removal. For example, constraint $x < y$ is the justification of the removal of value 1 from $y$'s domain. Figure 6 (2a) shows in grey the value removals that become questionable during the first step of arc consistency filtering at time 2. Because constraint $x < y$ has been removed and because this constraint is the justification of the removal of value 1 from $y$'s domain, this removal becomes questionable and this value must be restored. Note that, because the same constraint is not the justification of the removal of value 2 from $y$'s domain, this removal is not questionable. Then, because value 1 has been restored in $y$'s domain and because constraint $y = t$ is the justification of the removals of value 1 and 2 from $t$'s domain, both removals become questionable. Because value 1 is supported by the restored value, it must be restored.

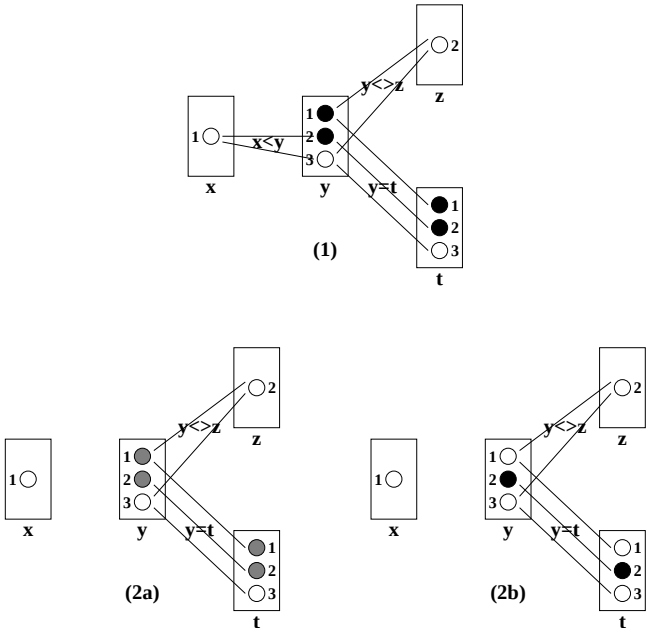


*Figure 5.* Graph-based method: (1) result of arc consistency filtering at time 1, (2a) questionable values during the first step of arc consistency filtering at time 2, (2b) result of the first and second steps of arc consistency filtering at time 2.

But, because value 2 is not supported by the restored value, it can remain removed. As with the previous graph-based method, Figure 6 (2b) shows in black the values that remain removed at the end of this first step and the result of the second step of arc consistency filtering at time 2. Note that, by using this justification-based method, not all the values that are removed by arc consistency filtering at time 1 become questionable during the first step of arc consistency filtering at time 2 (for example, the removal of value 2 from $y$'s domain is never questioned). Note also that this method needs to determine whether a questionable value is supported by another value, in order to determine whether it must be restored. This may result in costly constraint checks, but in a lower number than with the previous graph-based method, thanks to justifications.

Figure 7 (1) shows the result of arc consistency filtering at time 1, using an explanation-based method, like the one presented in (Debruyne et al., 2003), which associates with each removed value a set of constraints that imply this removal. For example, $\{y = t, x < y\}$ is an explanation of the removal of value 1 from $t$'s domain. Figure 7 (2a) shows in grey the value removals that become questionable during the

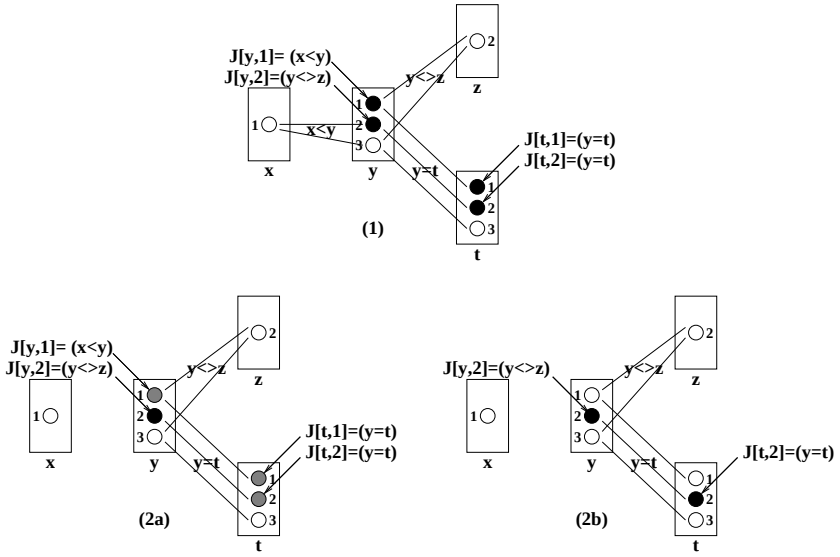


*Figure 6.* Justification-based method: (1) result of arc consistency filtering at time 1, (2a) questionable values during the first step of arc consistency filtering at time 2, (2b) result of the first and second steps of arc consistency filtering at time 2.

first step of arc consistency filtering at time 2. Because the removed constraint $x < y$ appears in the explanation of the removal of value 1 from $y$'s domain and $t$'s domain, both removals become questionable and associated values must be restored. Note that, contrary to the previous graph and justification-based methods, this method needs no restoration propagation in the constraint graph and that all the values that are associated with questionable removals must be restored. As with the previous graph and justification-based methods, Figure 7 (2b) shows in black the values that remain removed at the end of this first step and the result of the second step of arc consistency filtering at time 2. Note that, by using this explanation-based method, still fewer values that are removed by arc consistency filtering at time 1 become questionable during the first step of arc consistency filtering at time 2 (the removal of value 2 from $y$'s and $t$'s domains is never questioned). Note also that this method needs no constraint checks to determine whether a value must be restored[1].

[1] Note that this is also the case with the justification-based method $DnAC4$ (Bessière, 1991). Because it is based on the $AC4$ algorithm, all the constraint checks are performed in a preprocessing step and their results recorded in sets of supported values.

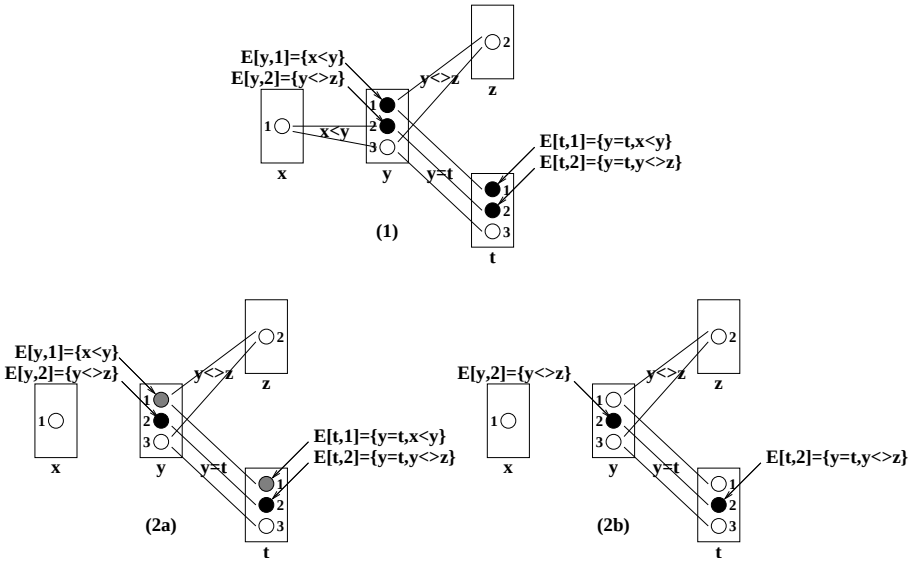


*Figure 7.* Explanation-based method: (1) result of arc consistency filtering at time 1, (2a) questionable values during the first step of arc consistency filtering at time 2, (2b) result of the first and second steps of arc consistency filtering at time 2.

### 5.6. Possible combinations

Because changes may combine problem restrictions and relaxations, combining solution and reasoning reuse techniques is obviously desirable.

This is for example what is done in (Hentenryck and Provost, 1991). In case of problem restriction, the previous solution is used as a heuristics to guide the new search (solution reuse). Moreover, the fact that no solution exists for the new problem before the previous solution, by using the same variable and value orderings, is used to limit the new search (reasoning reuse).

It can be also observed that search methods like *dynamic backtracking* (Ginsberg, 1993), and, more generally, local searches in the space of partial locally consistent assignments, like *decision repair* (Jussien and Lhomme, 2002), allow partial locally consistent assignments (potential parts of solutions) as well as value removals (reasoning results) to be reused in case of any change in the problem definition. See for example (Verfaillie and Schiex, 1994a; Jussien et al., 2000).

### 5.7. Complexity issues

From the complexity point of view, it must be observed that methods that reuse previous solutions and reasoning, on the one hand, and methods that solve the new problem from scratch after any change, on the other hand, both have the same *worst-case time complexity*: it may be necessary to destroy the whole previous solution to build a solution for the new problem and it is possible that all the previously implied constraints be invalid in the new problem.

Concerning the *worst-case space complexity* of the justification and explanation-based methods, it must be observed that the size of a justification or explanation is limited by the number of constraints and that only one justification or explanation is associated with every implied constraint. In such conditions, if the number of possible implied constraints is polynomial, the required space remains polynomial. This is for example the case with arc consistency, because possible implied constraints are only value removals, whose number is polynomial. This is more generally the case with any limited form of local consistency, such as $(i, j)$-consistency. Such a situation differs from the one of ATMS systems (de Kleer, 1986) which aim at producing all the possible implied constraints, along with all their possible explanations.

Experiments show that reusing previous solutions and reasoning is generally profitable, even in case of large changes, apart from solving hard instances at the frontier between consistency and inconsistency, possibly because small changes may have in that case a strong impact on the nature of the instance to solve. Such a situation suggests a strategy which would consist in reusing first previous solutions and reasoning for a short time, and restarting then from scratch in case of failure of the first solving method.

A synthesis of some of these reactive methods for dynamic CSPs, with experimental results, can be found in French in (Verfaillie and Schiex, 1995).

### 5.8. Search for a minimum change

It has been observed that solution reuse techniques tend to favour solution stability, i.e. the second of the above requirements (see Section 3) (Verfaillie and Schiex, 1995; Elkyari et al., 2004). However, this objective is not explicit and there is no guarantee about the result.

This is why some authors proposed methods whose objective is explicitly to minimise the distance between the previous solution and the new one. See for example (Bellicha, 1993; Ran et al., 2002) for dynamic CSP with a *Hamming* distance, and (Sakkout and Wallace, 2000) for dynamic scheduling problems with a distance which is defined as the

sum of the shifts in the temporal variables (start and end times of the activities).

## 6. Proactive approaches

### 6.1. Basic principles

Unlike reactive approaches which use no knowledge of the possible changes, proactive approaches use all the knowledge they may have about them in order to make decisions that will resist as much as possible these possible changes. These approaches can be split into two main families according to the nature of the solutions they produce, either *robust* or *flexible*:

1. given a model of the possible changes (qualitative or quantitative, probabilistic or not), a *robust* solution is a solution of the current problem that has every chance to resist changes, i.e. to remain a solution in spite of these changes;

2. a *flexible* solution is anything (partial solution, complete solution, conditional solution, set of solutions, etc.) that, in case of change, can be easily modified to produce a solution of the new problem. Note the relationship between the production of flexible solutions and the solution reuse techniques presented in Section 5.2. The main difference is however that, with flexible solutions, flexibility lies in the result, whereas, with solution reuse, flexibility lies in the algorithms which are capable of producing a new solution from the previous one.

### 6.2. Producing robust solutions

The frameworks that have been proposed so far to define and to produce robust solutions differ mainly in the model of uncertainty they use (see Section 4).

In (Wallace and Freuder, 1998), a *qualitative model* of the possible changes (temporary loss of values in the domains of the variables or of combination of values in the constraints) is used to favour, via heuristics, solutions that involve the most robust values or combinations of values. As an example, let us consider the small instance in Figure 8, slightly different from the one in Figure 2 (one more value in $z$'s domain) and let us assume that changes may occur in $z$'s domain: any of the two values may be lost, but the loss of value 1 is more likely than the

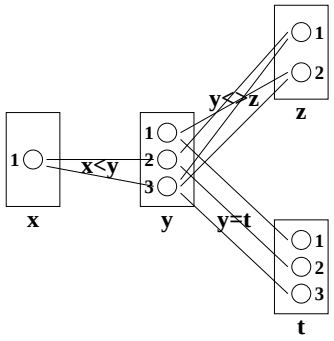


*Figure 8.* A second small instance, used to illustrate (Wallace and Freuder, 1998), (Fargier et al., 1996), and (Walsh, 2002).

one of value 2. In such conditions, value 2 is preferred and heuristics move the search towards solution $\{x = 1, y = 3, z = 2, t = 3\}$, among the three possible solutions.

In (Fargier et al., 1996), the model of the possible changes takes the form of *uncontrollable variables*, besides usual controllable ones, which may take any value in their domains. A usual objective is to produce a solution that is valid whatever the values taken by the uncontrollable variables. As an example, let us consider the same instance in Figure 8 and let us assume that variables $x$, $y$, and $t$ are controllable (we can decide upon their value), but that variable $z$ is uncontrollable (it may take any value: 1 or 2). In such conditions, $\{x = 1, y = 3, t = 3\}$ is a solution because it is consistent with any value of $z$, but $\{x = 1, y = 2, t = 2\}$ is not because it is not consistent with $z = 2$.

In (Walsh, 2002), a *probability distribution* is associated with the domain of each uncontrollable variable. A usual objective is thus to produce a solution whose probability of validity is maximum. As an example, let us consider the same instance in Figure 8, but let us assume now that variables $x$, $z$, and $t$ are controllable, but that variable $y$ is uncontrollable. In such conditions, there is no solution that is valid whatever the value taken by $y$. However, let us assume the following probability distribution over $y$'s domain: $P(y = 1) = 0.1, P(y = 2) = 0.7, P(y = 3) = 0.2$. In these conditions, some solutions are certainly not valid, such as $\{x = 1, z = 1, t = 1\}$ (inconsistency whatever the value taken by $y$). Other ones have a non null probability of validity such as $\{x = 1, z = 2, t = 3\}$ (consistency if $y = 3$; $P = 0.2$). But, the one whose probability of validity is maximum is $\{x = 1, z = 1, t = 2\}$ (consistency if $y = 2$; $P = 0.7$).

In (Fargier and Lang, 1993), the model of the possible changes takes the form of a *probability of existence* associated with each constraint.

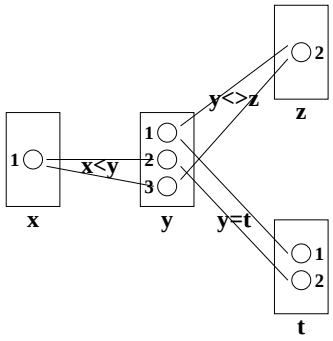


*Figure 9.* A third small instance, used to illustrate (Fargier and Lang, 1993) and (Fowler and Brown, 2000).

A usual objective is, as previously, to produce a solution whose probability of validity is maximum. As an example, let us consider the small instance in Figure 9, slightly different from the one in Figure 8 (one less value in $z$'s and $t$'s domains) and let us assume the following independent probabilities of existence associated with each constraint: $P(x < y) = 0.2, P(y \neq z) = 1, P(y = t) = 0.6$ (the constraint $y \neq z$ is certain; the other ones are uncertain). In these conditions, some solutions are certainly not valid, such as $\{x = 1, y = 2, z = 2, t = 1\}$ (constraint $y \neq z$ violated). Other ones have a non null probability of validity such as $\{x = 1, y = 1, z = 2, t = 2\}$ (constraints $x < y$ and $y = t$ violated; $P = (1 - 0.2) \cdot (1 - 0.6) = 0.32$). But, the one whose probability of validity is maximum is $\{x = 1, y = 1, z = 2, t = 1\}$ (constraint $x < y$ violated; $P = 1 - 0.2 = 0.8$).

In (Fowler and Brown, 2000), the model of the possible changes takes the form of a *probability of addition* to the current problem associated with each possible additional variable. Moreover, each variable may be assigned or not and a gain is associated with it assignment. The objective is to produce an assignment of the current variables that maximises the expected value of its extension to the additional variables. As an example, let us consider the same instance in Figure 9 and let us assume that variables $y$ and $z$ are present in the current problem, but that variables $x$ and $t$ are not. They may be added, $x$ with a probability of 0.8 and $t$ with a probability of 0.6. Moreover, let us assume that the gains associated with the assignment of variables $x$, $y$, $z$ and $t$ are respectively equal to 10, 20, 5, and 10. In such conditions, the expected gain associated for example with the assignment $\{y = 1, z = 2\}$ is equal to $20 + 5 + 0.6 \cdot 10 = 31$, because it will be possible to assign $t$, but not possible to assign $x$. In fact, the optimal assignment is $\{y = 2\}$ ($z$ not

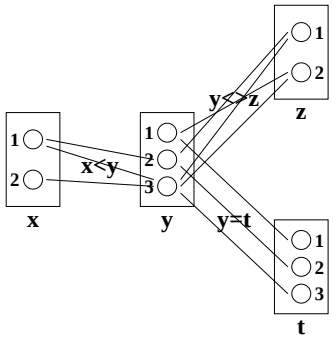


*Figure 10.* A fourth small instance, used to illustrate (Lesaint, 1993), (Amilhastre et al., 2002), and (Hebrard et al., 2004).

assigned) with an expected gain equal to $20 + 0.8 \cdot 10 + 0.6 \cdot 10 = 34$, because it will be possible to assign both $x$ and $t$.

All these approaches tend to favour solution robustness, i.e. the first of the above requirements (see Section 3). The reader may have a look at the references for the specific solving methods that are proposed in each of these uncertainty modelling frameworks.

Note the existence of specific approaches in related domains such as scheduling to build robust solutions in spite of uncertainty about actual activity durations. See for example (Davenport et al., 2001; Branke and Mattfeld, 2002).

### 6.3. Producing flexible solutions

The approaches that have been proposed so far to produce flexible solutions are somewhat disparate.

First, the notion of *value interchangeability*, introduced in (Freuder, 1991), can be considered as a basis for the production of flexible solutions.

In (Lesaint, 1993), a set of solutions, which takes the form of a *Cartesian product* of variable sub-domains, is considered as a flexible solution, because it will resist changes more likely than isolated solutions will. As an example, let us consider the small instance in Figure 10, slightly different from the one in Figure 2 (one more value in $x$'s and $z$'s domains). This instance has five solutions. However, we can observe that the set of solutions $\{x \in \{1,2\}, y = 3, z \in \{1,2\}, t = 3\}$, which compactly represents four solutions, is more flexible than the isolated solution $\{x = 1, y = 2, z = 1, t = 2\}$, because it will more likely resist changes in $x$'s and $z$'s domains, although it will not resist changes in $y$'s and $t$'s domains.

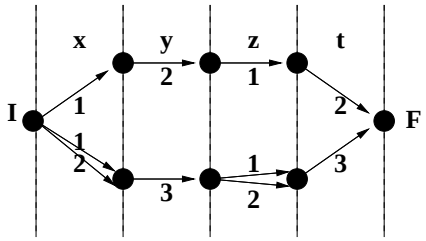


*Figure 11.* Non deterministic automaton associated with the instance in Figure 10.

In (Jégou, 1994; Amilhastre et al., 2002), a conditional solution, which takes the form of a *binary decision diagram* (BDD) (Bryant, 1986) or of an *automaton* (Vempaty, 1992), is used to compactly represent the set of the solutions of a dynamic CSP. As an example, let us consider the same instance in Figure 10. The non deterministic automaton showed in Figure 11 is a compact representation of the set of solutions of this instance, built by using the variable ordering $\{x, y, z, t\}$: any solution can be obtained by following any path $P$ from the initial state $I$ to the final state $F$ and by assigning at each stage the variable associated with this stage with the value associated with the arc of $P$ at this stage. For example, the solution $\{x = 2, y = 3, z = 2, t = 3\}$ is obtained by following systematically the arcs that are the lowest in the figure. Once built offline, such a structure can be used to react quickly online to any change in the domains resulting either from value removals or addings, or from variable assignments.

In (Ginsberg et al., 1998; Hebrard et al., 2004), a *supermodel* or *supersolution* $s$ has the property to remain a solution in case of any loss of their value in $s$ by a limited number $a$ of variables if we accept to change the value in $s$ of these $a$ variables and of a limited number $b$ of other variables. For example, an $(1, 1)$-supersolution is a solution $s$ such that, if we lose the value in $s$ of only one variable, we can recover a solution by changing this value and the value in $s$ of only one other variable. Let us consider the same instance in Figure 10. As previously noted, it has five solutions. However, we can observe that the solution $\{x = 1, y = 3, z = 1, t = 3\}$ is a $(1, 1)$-supersolution (no change for the other variables if the value 1 is lost in $x$'s domain, one change if the value 3 is lost in $y$'s domain, no change if the value 1 is lost in $z$'s domain, and one change if the value 3 is lost in $t$'s domain), whereas the other ones are $(1, 2)$ or $(1, 3)$-supersolutions. It will be consequently preferred.

Each one in its own way, all these methods tend to favour solution robustness and stability, as well as search efficiency, i.e. the first three of the above requirements (see Section 3). As with the methods presented in the previous section, the reader may have a look at the references for the specific solving methods that are proposed with each of these flexibility frameworks.

Note also specific approaches in the scheduling domain such as (Drummond et al., 1994), which presents the offline computing of a *contingent schedule* that anticipates the most probable breaks resulting from uncertainty about actual activity durations.

### 6.4. Possible combinations

Both qualities are not incompatible and the production of solutions that are at the same time robust and flexible, that have every chance to resist changes and can be easily adapted when they did not resist, is obviously a desirable objective.

A survey of the techniques that are used in the scheduling domain to deal with uncertainty can be found in (Davenport and Beck, 2000).

### 6.5. Complexity issues

Proactive methods have not been as systematically experimented as reactive methods have been. Moreover, they are still very diverse in terms of ambition, modelling features, and possible requests.

We can however say that adding to the problem definition information to be taken into account about the possible changes obviously does not decrease solving complexity. To be more precise, we can observe that, as SAT is the standard of constraint decision problems in certain environments and is *NP*-complete, QBF (*Quantified Boolean Formulas*), which is SAT with a free alternation of existential and universal quantifiers and is *Pspace*-complete, can be viewed as the standard of sequential constraint decision problems in uncertain environments. This provides us with an upper-bound in terms of *worst-case time complexity* for proactive methods. To go on with bad news, we must note that the *worst-case space complexity* of a representation of the set of solutions of a CSP as an automaton (see Section 6.3) is an exponential function of the instance size.

## 7. Research directions

In this section we consider the research directions that are, in our opinion, the most relevant if we want to deal more satisfactorily with uncertainties and changes in constraint solving.

### 7.1. A global requirement-oriented study

First, we think that studies about uncertainties and changes in constraint solving generally focused on very specific aspects, such as an efficient way of maintaining solutions or arc consistency results, or the way of producing stable or robust solutions. Although all these studies are useful, we think that a global study of what becomes constraint solving in uncertain and dynamic contexts (main requirements, main technical challenges, and most promising technical approaches) is still missing now. Such a study might lead us to discover that the main problems may not be where we thought they were and to reorient our research accordingly.

### 7.2. Incremental and decremental constraint propagation

It is however clear that, given that constraint propagation is the main reasoning tool in constraint solving, efficient incremental and decremental constraint propagation mechanisms are necessary if we want to deal with dynamic environments. This has been done for many years for the incremental side and remains to be done completely for the decremental side, by incorporating justification and explanation mechanisms as standards in constraint solving tools. See for example (Jussien and Barichard, 2000).

### 7.3. Handling controllable and uncontrollable variables

For the moment, constraint solving tools handle only two types of variables: those that represent possible decisions of an agent and those that represent possible values of an optimisation criterion. They do not handle correctly another type of variable: those that represent the possible *decisions* of the external world (state of the physical system and environment, real effect of decisions, information or requests from users and other entities, etc.), because these variables are uncontrollable: we cannot decide about them; we only can consider their possible values. Extending constraint solving mechanisms to handle uncontrollable as well as controllable variables is certainly one of the most important challenges of the next years and the key to deal correctly with uncer-

tainty. See for example (Bordeaux and Montfroy, 2002) for an extension of arc consistency to universally quantified variables.

### 7.4. Towards proactive and reactive methods

Roughly speaking, reactive methods based on solution reuse tend to favour search efficiency and solution stability, whereas reactive ones based on reasoning reuse tend to favour only search efficiency. Proactive methods based on the production of robust solutions obviously tend to favour solution robustness, whereas proactive ones based on the production of flexible solutions tend to favour solution robustness and stability, as well as search efficiency. All of them try to maintain solution consistency and optimality. If we want to satisfy as well as possible the four requirements listed in Section 3, it might be interesting to consider reactive-proactive methods that would anticipate possible future changes as far as information is available and would adapt themselves when changes occur (changes in the real world or in the information about possible future changes).

## 8. Connections with related topics

In this section we consider other modelling and reasoning frameworks that have something to do with the management of uncertainties and changes in constraint solving and that could be used, either as a source of inspiration for an extension of the CSP framework, or as a piece to combine with the CSP framework.

### 8.1. Stochastic satisfiability

The *stochastic satisfiability problem* (SSAT) (Littman et al., 2001) is an extension of the *satisfiability problem* (SAT). Whereas SAT considers only existential variables, SSAT allows any combination (any sequence) of existential, universal, and random variables, with a probability associated with the value of each random variable and an independence assumption between random variables and between random and non random variables. Such combinations allow for example decision problems under uncertainty, sequential or not, to be modelled. The *stochastic constraint satisfaction problem* (SCSP) (Walsh, 2002) does the same thing starting from the CSP framework. Although basic frameworks have been defined and first algorithms have been proposed (Manandhar et al., 2003), much work remains to be done, on the one hand, to propose more efficient and scalable algorithms and, on the

other hand, to relax the assumption of independence between variables (see Section 8.2).

### 8.2. Bayesian networks

*Bayesian networks* (Pearl, 1988; Neapolitan, 1990) have in common with *constraint networks* to consider finite domain variables and relations between variables. In constraint networks, these relations express allowed and forbidden combinations of values. In *valued constraint networks* (VCSP) (Schiex et al., 1995), they express costs of combinations of values. In Bayesian networks, they express conditional probabilities between the value of a variable $v$ and the values of its parent variables (those that may have an influence on $v$). The combination of constraint and Bayesian networks (see for example (Dechter and Larkin, 2001; Pralet et al., 2004)) may be the key to relax the assumption of independence between variables in the SSAT and SCSP frameworks and to deal with decision problems under constraints and uncertainties.

### 8.3. Markov decision processes

*Markov decision processes* (MDP) (Puterman, 1994) are widely used to model and solve sequential decision problems under uncertainty. Although efficient algorithms, exact as well as approximate, have already been proposed to compute optimal policies, MDP suffer from an extensive representation of states, actions, and transition probabilities. Problems become quickly unmanageable as soon as we want to model systems for which a significant number of variables is necessary to represent possible states and actions. More compact representations such as those that are used in constraint and Bayesian networks (see Section 8.3) seem to be unavoidable. See for example (Boutilier et al., 2000).

### 8.4. Temporal reasoning with uncertainty

In the domain of planning, scheduling, and temporal reasoning, a usual strategy to deal with uncertainty, referred to as *least commitment* strategy, consists in deciding about some crucial choices (for example, the activity selection and sequencing) and letting another process (for example execution control) make the remaining decisions (for example the exact starting time of the selected and sequenced activities) according to information coming from actual execution. In such a situation, the problem is to be sure that the remaining decisions will be able to be made consistently. In the STPU framework (*simple temporal problem with uncertainty* (Vidal and Fargier, 1999), an extension of

the STP framework (*simple temporal problem*) (Dechter et al., 1991) to deal with duration uncertainties, notions of *controllability* (Vidal and Fargier, 1999; Morris et al., 2001) and *dispatchability* (Muscettola et al., 1998a; Morris and Muscettola, 2000; Wallace and Freuder, 2005) and associated algorithms have been defined to offer such a guarantee. It would be interesting to extend such notions and algorithms to the more general CSP framework, by connecting STPU and MCSP.

## Acknowledgements

We would like to thank the anonymous reviewers and all the people who helped us clarifying our ideas and finally writing and improving this survey, particularly Romuald Debruyne, Simon de Givry, François Laburthe, Michel Lemaître, David Lesaint, Roland Prajoux, Thomas Schiex, and Thierry Vidal.

## References

Alami, R., R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand: 1998, 'An Architecture for Autonomy'. *The International Journal of Robotics Research* **17**(4), 315–337.

Amilhastre, J., H. Fargier, and P. Marquis: 2002, 'Consistency Restoration and Explanations in Dynamic CSP: Application to Configuration'. *Artificial Intelligence* **135**(1), 199–234.

Bellicha, A.: 1993, 'Maintenance of Solution in a Dynamic Constraint Satisfaction Problem'. In: *Proc. of Applications of Artificial Intelligence in Engineering VIII*. Toulouse, France, pp. 261–274.

Berlandier, P. and B. Neveu: 1994, 'Maintaining Arc Consistency through Constraint Retraction'. In: *Proc. of the 6th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-94)*. New Orleans, LA, USA.

Bessière, C.: 1991, 'Arc-Consistency in Dynamic Constraint Satisfaction Problems'. In: *Proc. of the 9th National Conference on Artificial Intelligence (AAAI-91)*. Anaheim, CA, USA, pp. 221–226.

Bessière, C.: 1992, 'Arc-Consistency for Non-Binary Dynamic CSPs'. In: *Proc. of the 10th European Conference on Artificial Intelligence (ECAI-92)*. Vienna, Austria, pp. 23–27.

Bistarelli, S., U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, and H. Fargier: 1999, 'Semiring-Based CSPs and Valued CSPs: Frameworks, Properties and Comparison'. *Constraints* **4**(3), 199–240.

Bordeaux, L. and E. Montfroy: 2002, 'Beyond NP: Arc-consistency for Quantified Constraints'. In: *Proc. of the 8th International Conference on Principles and Practice of Constraint Programming (CP-02)*. Ithaca, New York, USA.

Borning, A. and B. Freeman-Benson: 1998, 'Ultraviolet: A Constraint Satisfaction Algorithm for Interactive Graphics'. *Constraints* **3**(1), 9–32.

Boutilier, C., R. Dearden, and M. Goldszmidt: 2000, 'Stochastic Dynamic Programming with Factored Representations'. *Artificial Intelligence* **121**(1-2), 49–107.

Bowen, J. and G. Bahler: 1991, 'Conditional Existence of Variables in Generalized Constraint Networks'. In: *Proc. of the 9th National Conference on Artificial Intelligence (AAAI-91)*. Anaheim, CA, USA.

Branke, J. and D. Mattfeld: 2002, 'Anticipatory Scheduling for Dynamic Job Shop Problems'. In: *Proc. of the AIPS-02 Workshop on "Online Planning and Scheduling"*. Toulouse, France.

Bryant, R.: 1986, 'Graph-Based Algorithms for Boolean Function Manipulation'. *IEEE Transactions on Computers* **C-35**(8), 677–691.

Chien, S., R. Knight, A. Stechert, R.Sherwood, and G. Rabideau: 2000, 'Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling'. In: *Proc. of the 5th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-00)*. Breckenridge, CO, USA.

Davenport, A. and C. Beck: 2000, 'A Survey of Techniques for Scheduling with Uncertainty'. http://4c.ucc.ie/ jcb/publications.html.

Davenport, A., C. Gefflot, and C. Beck: 2001, 'Slack-based Techniques for Robust Schedules'. In: *Proc. of the 6th European Conference on Planning (ECP-01)*. Toledo, Spain.

de Kleer, J.: 1986, 'An Assumption-based Truth Maintenance System'. *Artificial Intelligence* **28**, 127–162.

Debruyne, R.: 1995, 'Les algorithmes d'arc-consistance dans les CSP dynamiques'. *Revue d'Intelligence Artificielle* **9**(3), 239–267.

Debruyne, R.: 1996, 'Arc-Consistency in Dynamic CSPs is no more Prohibitive'. In: *Proc. of the 8th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-96)*. Toulouse, France, pp. 239–267.

Debruyne, R., G. Ferrand, N. Jussien, W. Lesaint, S. Ouis, and A. Tessier: 2003, 'Correctness of Constraint Retraction Algorithms'. In: *Proc. of the 16th International Florida Artificial Intelligence Research Society Conference (FLAIRS-03)*. St. Augustine, FL, USA.

Dechter, R.: 1992, 'Constraint Networks'. In: S. Shapiro (ed.): *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, pp. 276–285.

Dechter, R. and A. Dechter: 1988, 'Belief Maintenance in Dynamic Constraint Networks'. In: *Proc. of the 7th National Conference on Artificial Intelligence (AAAI-88)*. St. Paul, MN, USA, pp. 37–42.

Dechter, R. and D. Larkin: 2001, 'Hybrid Processing of Beliefs and Constraints'. In: *Proc. of the 17th International Conference on Uncertainty in Artificial Intelligence (UAI-01)*. Seattle, WA, USA, pp. 112–119.

Dechter, R., I. Meiry, and J. Pearl: 1991, 'Temporal Constraint Networks'. *Artificial Intelligence* **49**, 61–95.

Deransart, P., M. Hermenegildo, and J. Malluszynski (eds.): 2000, *Analysis and Visualisation Tools for Constraint Programming (LNCS 1870)*. Springer Verlag.

Doyle, J.: 1979, 'A Truth Maintenance System'. *Artificial Intelligence* **12**, 231–272.

Drummond, M., J. Bresina, and K. Swanson: 1994, 'Just-In-Case Scheduling'. In: *Proc. of the 12th National Conference on Artificial Intelligence (AAAI-94)*. Seattle, WA, USA, pp. 1098–1104.

Elkyari, A., C. Guéret, and N. Jussien: 2004, 'Stable Solutions for Dynamic Project Scheduling Problems'. In: *Proc. of the International Workshop on Project Management and Scheduling (PMS-04)*. Nancy, France, pp. 380–384.

Fages, F., J. Fowler, and T. Sola: 1998, 'Experiments in Reactive Constraint Logic Programming'. *Journal of Logic Programming* **37**(1-3), 185–212.

Faltings, B. and S. Macho-Gonzalez: 2002, 'Open Constraint Satisfaction'. In: *Proc. of the 8th International Conference on Principles and Practice of Constraint Programming (CP-02)*. Ithaca, New York, USA, pp. 356–370.

Fargier, H. and J. Lang: 1993, 'Uncertainty in Constraint Satisfaction Problems: A Probabilistic Approach'. In: *Proc. of the European Conference on Symbolic and Quantitavive Approaches of Reasoning under Uncertainty (ECSQARU-93)*. Grenade, Spain, pp. 97–104.

Fargier, H., J. Lang, and T. Schiex: 1996, 'Mixed Constraint Satisfaction: a Framework for Decision Problems under Incomplete Knowledge'. In: *Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96)*. Portland, OR, USA, pp. 175–180.

Fowler, D. and K. Brown: 2000, 'Branching Constraint Satisfaction Problems for Solutions Robust under Likely Changes'. In: *Proc. of the 6th International Conference on Principles and Practice of Constraint Programming (CP-00)*. Singapore.

Fowler, D. and K. Brown: 2003, 'Branching Constraint Satisfaction Problems and Markov Decision Problems Compared'. *Annals of Operations Research* **118**(1-4), 85–100.

Freeman-Benson, B., J. Maloney, and A. Borning: 1990, 'An Incremental Constraint Solver'. *Communications of the ACM* **33**(1), 54–63.

Freuder, E.: 1991, 'Eliminating Interchangeable Values in Constraint Satisfaction Problems'. In: *Proc. of the 9th National Conference on Artificial Intelligence (AAAI-91)*. Anaheim, CA, USA, pp. 227–233.

Gelle, E. and B. Faltings: 2003, 'Solving Mixed and Conditional Constraint Satisfaction Problems'. *Constraints* **8**(2), 107–141.

Georget, Y., P. Codognet, and F. Rossi: 1999, 'Constraint Retraction in CLP(FD): Formal Framework and Performance Results'. *Constraints* **4**(1), 5–42.

Ginsberg, M.: 1993, 'Dynamic Backtracking'. *Journal of Artificial Intelligence Research* **1**, 25–46.

Ginsberg, M., A. Parkes, and A. Roy: 1998, 'Supermodels and Robustness'. In: *Proc. of the 15th National Conference on Artificial Intelligence (AAAI-98)*. Madison, WI, USA, pp. 334–339.

Harvey, W. and M. Ginsberg: 1995, 'Limited Discrepancy Search'. In: *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*. Montréal, Canada, pp. 607–613.

Hebrard, E., B. Hnich, and T. Walsh: 2004, 'Super Solutions in Constraint Programming'. In: *Proc. of the International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimisation Problems (CP-AI-OR-04)*. Nice, France, pp. 157–172.

Hentenryck, P. V. and T. L. Provost: 1991, 'Incremental Search in Constraint Logic Programming'. *New Generation Computing* **9**, 257–275.

Jégou, P.: 1994, 'A Logical Approach to Solve Dynamic CSPs: Preliminary Report'. In: *Proc. of the ECAI94 Workshop on "Constraint Satisfaction Issues Raised by Practical Applications"*. Amsterdam, The Netherlands.

Jussien, N. and V. Barichard: 2000, 'The PaLM System: Explanation-based Constraint Programming'. In: *Proc. of the CP-00 TRICS Workshop on "Techniques foR Implementing Constraint programming Systems"*. Singapore, pp. 118–133.

Jussien, N., R. Debruyne, and P. Boizumault: 2000, 'Maintaining Arc-Consistency within Dynamic Backtracking'. In: *Proc. of the 6th International Conference*

*on Principles and Practice of Constraint Programming (CP-00)*. Singapore, pp. 249–261.

Jussien, N. and O. Lhomme: 2002, 'Local Search with Constraint Propagation and Conflict-based Heuristics'. *Artificial Intelligence* **139**, 21–45.

Kramer, L. and S. Smith: 2003, 'Maximizing Flexibility: A Retraction Heuristic for Oversubscribed Scheduling Problems'. In: *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*. Acapulco, Mexico.

Lamma, E., P. Mello, M. Milano, R. Cucchiara, M. Gavanelli, and M. Piccardi: 1999, 'Constraint Propagation and Value Acquisition: Why we should do it Interactively'. In: *Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*. Stockholm, Sweden.

Lesaint, D.: 1993, 'Specific solution sets for Constraint Satisfaction Problems'. In: *Proc. of the 13th International Conference on Artificial Intelligence, Expert Systems and Natural Language (Avignon-93)*. Avignon, France.

Littman, M., S. Majercik, and T. Pitassi: 2001, 'Stochastic Boolean Satisfiability'. *Journal of Automated Reasoning* **27**(3), 251–296.

Lottaz, C., D. Clément, B. Faltings, and I. Smith: 1999, 'Constraint-Based Support for Collaboration in Design and Construction'. *Journal of Computing in Civil Engineering* **13**(1), 23–35.

Mackworth, A.: 1992, 'Constraint Satisfaction'. In: S. Shapiro (ed.): *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, pp. 285–293.

Manandhar, S., A. Tarim, and T. Walsh: 2003, 'Scenario-based Stochastic Constraint Programming'. In: *Proc. of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*. Acapulco, Mexico.

Miguel, I. and Q. Shen: 2003, 'Fuzzy rrDFCSP and Planning'. *Artificial Intelligence* **148**(1-2), 11–52.

Minton, S., M. Johnston, A. Philips, and P. Laird: 1992, 'Minimizing Conflicts: a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems'. *Artificial Intelligence* **58**, 160–205.

Mittal, S. and B. Falkenhainer: 1990, 'Dynamic Constraint Satisfaction Problems'. In: *Proc. of the 8th National Conference on Artificial Intelligence (AAAI-90)*. Boston, MA, USA, pp. 25–32.

Montanari, U. and F. Rossi: 1997, 'Constraint Solving and Programming: What Next?'. *Constraints* **2**(1), 87–91.

Morris, P. and N. Muscettola: 2000, 'Execution of Temporal Plans with Uncertainty'. In: *Proc. of the 17th National Conference on Artificial Intelligence (AAAI-00)*. Austin, TX, USA, pp. 491–496.

Morris, P., N. Muscettola, and T. Vidal: 2001, 'Dynamic Control of Plans with Temporal Uncertainty'. In: *Proc. of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*. Seattle, WA, USA, pp. 494–499.

Muscettola, N., P. Morris, and I. Tsamardinos: 1998a, 'Reformulating Temporal Plans for Efficient Execution'. In: *Proc. of the 6th International Conference on the Principles of Knowledge Representation and Reasoning (KR-98)*. Trento, Italy, pp. 444–452.

Muscettola, N., P. Nayak, B. Pell, and B. Williams: 1998b, 'Remote Agent: To Boldly Go Where No AI System Has Gone Before'. *Artificial Intelligence* **103**(1-2), 5–48.

Neapolitan, R.: 1990, *Probabilistic Reasoning in Expert Systems*. Wiley and Sons.

O'Sullivan, B.: 2002, 'Interactive Constraint-Aided Conceptual Design'. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing Journal (AIEDAM)* **16**(4).

Pearl, J.: 1988, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pralet, C., G. Verfaillie, and T. Schiex: 2004, 'Belief and Desire Networks for Answering Complex Queries'. In: *Proc. of the CP-04 Workshop on "Constraint Solving under Change and Uncertainty"*. Toronto, Canada.

Prosser, P.: 1989, 'A Reactive Scheduling Agent'. In: *Proc. of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*. Detroit, MI, USA, pp. 1004–1009.

Prosser, P., C. Conway, and C. Muller: 1992, 'A Distributed Constraint Maintenance System'. In: *Proc. of the 12th International Conference on Artificial Intelligence, Expert Systems and Natural Language (Avignon-92)*. Avignon, France, pp. 221–231.

Puterman, M.: 1994, *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Ran, Y., N. Roos, and J. van den Herik: 2002, 'Approaches to Find a Near-minimal Change Solution for Dynamic CSPs'. In: *Proc. of the 4th International Workshop on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR-02)*. Le Croisic, France.

Sabin, D. and E. Freuder: 1996, 'Configuration as Composite Constraint Satisfaction'. In: *Working Notes of the AAAI96 Fall Symposium on Configuration*.

Sabin, D. and E. Freuder: 1998, 'Detecting and Resolving Inconsistency and Redundancy in Conditional Constraint Satisfaction Problems'. In: *Proc. of the CP-98 Workshop on "Constraint Problem Reformulation"*. Pisa, Italia.

Sabin, D. and E. Freuder: 1999, 'Detecting and Resolving Inconsistency in Conditional Constraint Satisfaction Problems'. In: *Proc. of the AAAI-99 Workshop on "Configuration"*. Orlando, FL, USA, pp. 95–100.

Sabin, M., E. Freuder, and R. Wallace: 2003, 'Greater Efficiency for Conditional Constraint Satisfaction'. In: *Proc. of the 9th International Conference on Principles and Practice of Constraint Programming (CP-03)*. Cork, Ireland, pp. 649–663.

Sakkout, H. E. and M. Wallace: 2000, 'Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling'. *Constraints* **5**(4), 359–388.

Sannella, M., J. Maloney, B. Freeman-Benson, and A. Borning: 1993, 'Multi-way versus One-way Constraints in User Interfaces: Experience with the DeltaBlue Algorithm'. *Software: Practice and Experience* **23**(5), 529–566.

Schiex, T., H. Fargier, and G. Verfaillie: 1995, 'Valued Constraint Satisfaction Problems : Hard and Easy Problems'. In: *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*. Montréal, Canada, pp. 631–637.

Schiex, T. and G. Verfaillie: 1993, 'Nogood Recording for Static and Dynamic CSP'. In: *Proc. of the 5th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-93)*. Boston, MA, USA, pp. 48–55.

Smith, S.: 1995, 'Reactive Scheduling Systems'. In: D. Brown and W. Sherer (eds.): *Intelligent Scheduling Systems*. Kluwer.

Soininen, T., E. Gelle, and I. Niemela: 1999, 'A Fixpoint Definition of Dynamic Constraint Satisfaction'. In: *Proc. of the 5th International Conference on Principles and Practice of Constraint Programming (CP-99)*. Alexandria, VA, USA, pp. 419–433.

Trombettoni, G.: 1998, 'A Polynomial Time Local Propagation Algorithm for General Dataflow Constraint Problems'. In: *Proc. of the 4th International Conference*

*on Principles and Practice of Constraint Programming (CP-98)*. Pisa, Italia, pp. 432–446.

van Beek, P. and X. Chen: 1999, 'CPlan: A Constraint Programming Approach to Planning'. In: *Proc. of the 16th National Conference on Artificial Intelligence (AAAI-99)*. Orlando, FL, USA, pp. 585–590.

Vempaty, N.: 1992, 'Solving Constraint Satisfaction Problems Using Finite State Automata'. In: *Proc. of the 10th National Conference on Artificial Intelligence (AAAI-92)*. San Jose, CA, USA, pp. 453–458.

Verfaillie, G. and N. Jussien: 2003, 'Dynamic Constraint Solving: Handling Change and Uncertainty in Constraint Satisfaction'. http://www.emn.fr/jussien/CP03tutorial: slides of the tutorial given at CP-03 (9th International Conference on Principles and Practice of Constraint Programming, Kinsale, Ireland), with a commented bibliography.

Verfaillie, G. and T. Schiex: 1994a, 'Dynamic Backtracking for Dynamic Constraint Satisfaction Problems'. In: *Proc. of the ECAI-94 Workshop on "Constraint Satisfaction Issues Raised by Practical Applications"*. Amsterdam, The Netherlands.

Verfaillie, G. and T. Schiex: 1994b, 'Solution Reuse in Dynamic Constraint Satisfaction Problems'. In: *Proc. of the 12th National Conference on Artificial Intelligence (AAAI-94)*. Seattle, WA, USA, pp. 307–312.

Verfaillie, G. and T. Schiex: 1995, 'Maintien de solution dans les problèmes dynamiques de satisfaction de contraintes: bilan de quelques approches'. *Revue d'Intelligence Artificielle* **9**(3), 269–309.

Vidal, T. and H. Fargier: 1999, 'Handling Contingency in Temporal Constraint Networks: From Consistency to Controllabilities'. *Journal of Experimental and Theoretical Artificial Intelligence* **11**(1), 23–45.

Wallace, R. and E. Freuder: 1998, 'Stable Solutions for Dynamic Constraint Satisfaction Problems'. In: *Proc. of the 4th International Conference on Principles and Practice of Constraint Programming (CP-98)*. Pisa, Italia, pp. 447–461.

Wallace, R. and E. Freuder: 2005, 'Supporting Dispatchability in Schedules with Consumable Ressources'. *Journal of Scheduling* **8**, 7–23.

Walsh, T.: 2002, 'Stochastic Constraint Programming'. In: *Proc. of the 15th European Conference on Artificial Intelligence (ECAI-02)*. Lyon, France.

Yokoo, M., E. Durfee, T. Ishida, and K. Kuwabara: 1998, 'The Distributed Constraint Satisfaction Problem: Formalization and Algorithms'. *IEEE Transactions on Knowledge and Data Engineering* **10**(5), 673–685.

Zweden, M., B. Daun, E. Davis, and M. Deale: 1994, 'Scheduling and Rescheduling with Iterative Repair'. In: M. Zweden and M. Fox (eds.): *Intelligent Scheduling*. Morgan Kaufmann, pp. 241–256.