Handbook of Constraint Programming
Edited by F. Rossi, P. van Beek and T. Walsh
© 2006 Elsevier All rights reserved

Chapter 21

Uncertainty and Change

Kenneth N. Brown and Ian Miguel

Constraint Programming (CP) has proven to be a very successful technique for reasoning about assignment problems, as evidenced by the many applications described elsewhere in this book. Much of its success is due to the simple and elegant underlying formulation: describe the world in terms of decision variables that must be assigned values, place clear and explicit restrictions on the values that may be assigned simultaneously, and then find a set of assignments to all the variables that obeys those restrictions. Thus, CP makes two assumptions about the problems it tackles:

1. There is no *uncertainty* in the problem definition: each problem has a crisp and complete description.

2. Problems are not *dynamic*: they do not change between the initial description and the final execution of the solution.

Unfortunately, these two assumptions do not hold for many practical and important applications. For example, scheduling production in a factory is, in practice, fundamentally dynamic and uncertain: the full set of jobs to be scheduled is not known in advance, and continues to grow as existing jobs are being completed; machines break down; raw material is delivered late; employees become ill; jobs take longer than expected; or processes have inherently random aspects, and so some jobs may have to be repeated. Alternatively, in engineering or architectural design, the constraints themselves are not known with certainty — this may be because the designer is not aware of the detail of the constraints, or because the constraints are inherently vague — or may be changing because the designer is exploring the problem space, reformulating the problem as the consequences of each modelling decision become clearer.

Current constraint solving tools provide very little support for explicit reasoning about uncertain and dynamic problems. In many cases, an approximated deterministic and static model may suffice, and provides the user with enough information about the structure of the problem to make good enough decisions. In other cases, though, the user is required to re-formulate the problem repeatedly, in response to each change or to each discovery of more detail of the problem. What support should CP tools offer in those situations? For many problems, all that may be required is a sufficiently fast solver, reacting to the changes with new solutions, or producing many initial solutions to different formulations in the case of uncertainty. At other times, for dynamic problems, the new solutions should be as close as possible to the previous ones, to minimise the cost of change. More advanced methods should generate solutions that are robust to the likely changes, or that are sufficiently flexible to allow the changes to be accommodated. Particular attention should also be paid to time limits, since the dynamic changes may occur too quickly to allow exhaustive analysis — in that case, time-bounded or anytime reasoning is required.

In this chapter, we consider the uses and extensions of constraint programming for handling problems subject to change and uncertainty. We classify the research into two broad categories based on the problem type:

(i) uncertain problems, which require a single solution; and

(ii) Dynamically changing problems, which require multiple solution stages.

Within (ii), we consider three further sub-categories:

(ii-a) problems where the solver simply reacts each time the problems change;

(ii-b) problems where the solving process is adapted to record information about the problem structure, which can be used during the reaction phase; and

(ii-c) problems where the solver proactively searches for solutions that anticipate the expected changes.

We will begin by briefly reviewing the definitions of constraint satisfaction and optimisation problems, and presenting a small example problem which we will use throughout the chapter. We will then consider each of the categories and sub-categories in turn. Finally, we will conclude with a discussion of challenges for future research.

21.1 Background and Definitions

The finite-domain *constraint satisfaction problem* (CSP) consists of a triple $\langle X, D, C \rangle$, where X is a set of variables, D is a set of domains, and C is a set of constraints. Each $x_i \in X$ is associated with a finite domain $D_i \in D$ of potential values. An *assignment* to a variable x_i is the selection of a value v_i from its domain D_i. A constraint $c \in C$, constraining variables $x_i, \ldots, x_j$, specifies a subset of the Cartesian product $D_i \times \ldots \times D_j$ indicating mutually-compatible variable assignments. A tuple of values $v = (v_i, \ldots, v_j)$ *satisfies* a constraint c over $x_i, \ldots, x_j$ if $v \in c$. A *partial assignment* to a problem is a collection of assignments to a subset of the variables in the problem, and a complete assignment is an assignment for every variable. A *solution* is a complete assignment that satisfies all constraints. A *constrained optimisation problem* is a CSP with some objective function, which is to be optimised.

21.2 Example: Course Scheduling

To illustrate the various problems and techniques, we will use as a basis the following simple example (adapted from [22]) throughout the chapter. Consider the task of scheduling

Type

Days	i \ j	L(1)	P(2)	T(3)
	1	x_{11}	x_{12}	x_{13}
	2	x_{21}	x_{22}	x_{23}
	3	x_{31}	x_{23}	x_{33}

$$\forall i \; \sum_{j=1}^{3} x_{ij} \geq 2 \qquad \textit{sessions per day} \tag{21.1}$$

$$\forall j \; \sum_{i=1}^{3} x_{ij} \in \{1, 2, \ldots, 5\} \qquad \textit{no. of session type} \tag{21.2}$$

$$\sum_{i=1}^{3} \sum_{j=1}^{3} x_{ij} \in \{10, 11, 12\} \qquad \textit{total sessions} \tag{21.3}$$

Figure 21.1: Course Scheduling Problem

a short course over three days consisting of a number of lectures, practical sessions, and tutorial sessions. The constraints are that there must be at least two sessions a day and, over the three days, there must be between 1 and 5 of each type of session and between 10 and 12 sessions in total. This problem can be cast as a CSP by using 9 variables, x_{ij} with i and j in {1, 2, 3}, where i denotes the day and j the session type with 1 = lecture, 2 = practical, 3 = tutorial. Each variable has domain {0, 1, 2, 3, 4, 5} denoting the number of sessions of the corresponding type on a particular day. The constraints are expressed on these variables as presented in Figure 21.1. Figure 21.2 presents one possible solution to this problem in which there are two lectures, three practical and five tutorial sessions over the three days.

Type

Days	i \ j	L(1)	P(2)	T(3)
	1	1	0	1
	2	1	1	0
	3	0	2	4

Figure 21.2: A Solution to Course Scheduling Problem

21.3 Uncertain Problems

First we consider problems where a complete crisp description of the problem will not be revealed at all, and so we must produce a single initial solution that cannot be changed. In order to produce the solutions, we have to consider how the imprecision in the problem description is expressed. We consider three cases: *(i)* the problem itself is intrinsically imprecise — for example, where the price of a configuration must be 'cheap', where 'cheap'

$$\forall i \sum_{j=1}^{3} x_{ij} \geq 2 \qquad \textit{sessions per day}$$

$$\sum_{i=1}^{3}\sum_{j=1}^{3} x_{ij} \in \{10, 11, 12\} \qquad \textit{total sessions}$$

		Sums of Assignment Tuples					
		1	2	3	4	5	Otherwise
lectures	$\sum_{i=1}^{3} x_{i1}$	0.4	0.6	0.8	1.0	0.8	0
practicals	$\sum_{i=1}^{3} x_{i2}$	0.6	0.8	1.0	0.8	0.6	0
tutorials	$\sum_{i=1}^{3} x_{i3}$	0.6	0.8	1.0	0.8	0.6	0

Figure 21.3: The Fuzzy Course Scheduling Problem. Fuzzy constraints show satisfaction degrees for different possible assignment tuples.

is defined by a fuzzy membership function, *(ii)* we have a set of possible realisations of the problem, one of which will be the final version of the problem, and *(iii)* we have probability distributions over the full realisations — for example, a distribution over the values that might be available to us, or over the legal tuples in the constraints. Secondly, for *(ii)* and *(iii)*, we also consider problems where the description will eventually be revealed, but requires an instant response. In such cases, we can extend the techniques to include contingencies — families of solutions, one of which will be selected depending on the revealed problem.

21.3.1 Fuzzy Problems

Fuzzy constraint satisfaction [22] (see also Chapter 9) captures imprecision in the definition of a constraint by allowing constraints to be partially satisfied, as well as completely satisfied or completely unsatisfied. To continue the above example, a constraint specifying that an expression in certain cost variables must be "cheap", rather than being satisfied or violated, can be satisfied to a greater or lesser extent according to the assignments to the cost variables. This allows us to capture notions such as "fairly cheap" and "relatively expensive".

In a fuzzy constraint satisfaction problem, a constraint $c(x_i, \ldots, x_j)$ is represented by a fuzzy relation, which is in turn defined by a *membership function* that associates a degree of satisfaction in a totally ordered scale (usually [0, 1], with 0 and 1 representing complete violation and complete satisfaction respectively) with each tuple in $D_i \times \ldots \times D_j$. The conjunction of two fuzzy relations is usually interpreted as the minimum membership value assigned by either relation. To produce a satisfaction degree for a given partial or complete assignment, the conjunction operator is used to aggregate the satisfaction degrees of all constraints on the assigned variables. This allows us to rank different assignments and therefore search for optimal solutions to a fuzzy CSP.

To illustrate, we consider a fuzzy version of the course scheduling problem given in Figure 21.1. Professor A is to give the lectures in the course. She prefers to give four

Constraint	Assignment Sum	Sat Degree
Sessions per Day	2, 4, 4	1.0
Total Sessions	10	1.0
Lectures	3	0.8
Practicals	3	1.0
Tutorials	4	0.8
Overall Satisfaction Degree: 0.8		

		Type		
	i \ j	L(1)	P(2)	T(3)
Days	1	1	0	1
	2	1	1	0
	3	0	2	4

Constraint	Assignment Sum	Sat Degree
Sessions per Day	2, 4, 4	1.0
Total Sessions	10	1.0
Lectures	4	1.0
Practicals	3	1.0
Tutorials	3	1.0
Overall Satisfaction Degree: 1.0		

		Type		
	i \ j	L(1)	P(2)	T(3)
Days	1	1	0	1
	2	2	1	1
	3	1	2	1

Figure 21.4: Sub-optimal and Optimal Solutions to the Fuzzy Course Scheduling Problem

of these sessions. Dr B is organising the practical sessions, and he prefers to give about three of these. Finally, Dr C is responsible for the tutorial sessions, and also prefers that there should be about three of these. These preferences are captured in fuzzy constraints on the lecture, practical and tutorial session variables, as presented in Figure 21.3. Note that constraints on the number of sessions per day and the total number of sessions remain as hard constraints. Hard constraints are simple to represent with fuzzy constraints: the satisfaction degree of each assignment tuple is either 0 or 1.

Figure 21.4 presents two solutions to the fuzzy course scheduling problem. The first is the same as the solution to the crisp course scheduling problem given in Figure 21.2. This solution has satisfaction degree 0.8 because there are three lecture sessions (from Figure 21.2, the satisfaction degree of the constraint on the number of lectures is therefore 0.8) and three tutorial sessions (also satisfaction degree 0.8). Hence, the fuzzy conjunction of the satisfaction degrees of all the constraints is 0.8. The second solution has satisfaction degree 1.0 and is therefore optimal. The reader will be able to confirm that the satisfaction degree of each constraint is 1.0.

21.3.2 Problems with Possible Realisations

For problems with a set of possible realisations, we first need to consider the ways in which the problem definition could be incomplete — i.e. what is missing from the original description that will be revealed. Based on the definition in 21.1, this could be:

1. The complete set of variables is not known;

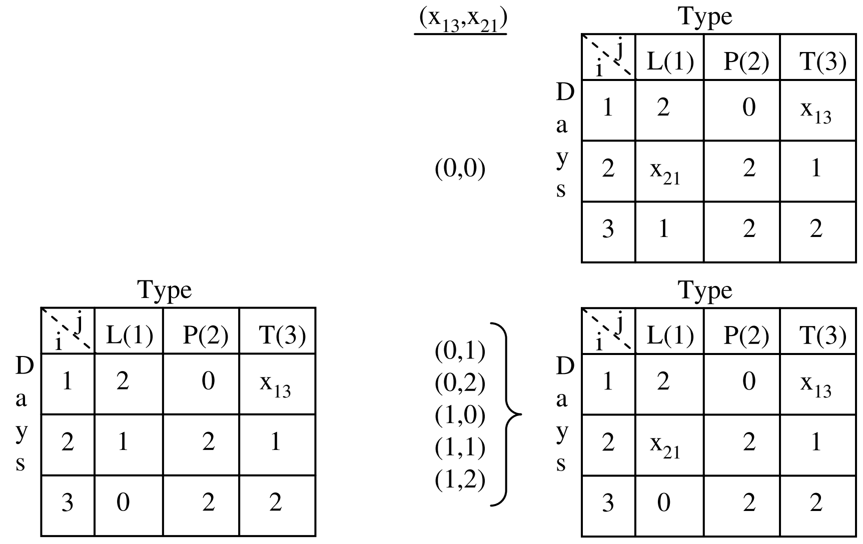
2. The domains of the variables are not completely specified; or
3. The constraints are not completely specified — either the full set of constraints is not known, or the individual constraints are not fully described.

In fact, we could reformulate the definition of a CSP so that only the constraints need to be specified explicitly (the domains would be unary constraints restricting values from a universal set, and the variables are implicitly defined to be those appearing anywhere in the constraint set), and thus formally we only need to consider uncertainty in the constraint set. In practice, the different types of uncertainty are treated separately, to model specific features of different application domains, and give rise to different formalisms and algorithms.

In *Mixed CSPs* [27], we model the case where some of the variables are not controlled by the solver, but will be assigned by some external source (which may be a user, another agent, later knowledge discovery, or a random process). Thus the variables of the problem are divided into two classes: controlled decision variables and uncontrollable parameters. The decision variables are normal CSP variables, but the parameters will be set by the external source (and thus essentially fix the domains of those variables to a singleton set). The possible realisations of the problem are then defined by the sets of possible values that the parameters may take. Constraints restrict the assignments of values to variables in the normal way. A *pure* decision is an assignment of values to all the decision variables, which should be a solution to one or more of the possible realisations. If there are no constraints on the realisations (i.e. the parameters are independent), then it is NP-complete to determine whether there exists a single pure decision which is a solution to all realisations in a binary mixed CSP. For cases where the true realisation will be revealed, a *conditional* decision associates different assignments of values to different realisations, and an *optimal* conditional decision has a solution for each possible realisation. Fargier *et al* [27] give an anytime algorithm for finding conditional decisions.

As an example, consider the course scheduling problem as before, but now we assume that the number of tutorials on day 1 (x_{13}) will be decided later (based on the availability of tutors). That is, the variable x_{13} becomes an uncontrollable parameter. Suppose we know that x_{13} can take one of two possible values, 0 or 1. Figure 21.5a shows a pure decision for all the other variables that satisfies both possible realisations. Suppose now that the number of lectures on day 2 (x_{21}) will also be fixed at a later date, and that the value of x_{21} may be 0, 1 or 2, independently of the value of x_{13}. There are now six possible realisations, based on the possible values of (x_{13}, x_{21}): $\{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2)\}$. No single pure decision is possible (since it will not be possible to satisfy the constraint on the total number of sessions); however, figure 21.5b shows an optimal conditional decision, by associating a decision with each possible realisation.

To cover problems with uncertain data, Yorke-Smith and Gervet define *Uncertain CSPs* [77], in which the constraints are uncertain — specifically, they use an algebraic representation of the constraints, with uncertainty over the coefficients. Their goal is to define the *certainty closure*, the set of all solutions to possible realisations of the constraints, and then to search for specific types of closure, including a *covering set*, which contains at least one solution for each realisation, or the *most robust solution*, which is a solution to the greatest number of realisations. Their suggested solution method is to transform the UCSP into a standard CSP, such that the set of all solutions to the CSP is equivalent to the desired closure of the UCSP.



(a) $x_{13} \in \{0,1\}$

Type

i \ j	L(1)	P(2)	T(3)
1	2	0	x_{13}
2	1	2	1
3	0	2	2

(b) $x_{13} \in \{0,1\}$, $x_{21} \in \{0,1,2\}$

(x_{13},x_{21}): (0,0)

Type

i \ j	L(1)	P(2)	T(3)
1	2	0	x_{13}
2	x_{21}	2	1
3	1	2	2

(x_{13},x_{21}): (0,1), (0,2), (1,0), (1,1), (1,2)

Type

i \ j	L(1)	P(2)	T(3)
1	2	0	x_{13}
2	x_{21}	2	1
3	0	2	2

Figure 21.5: *mixed* CSP: *(a)* a pure decision; *(b)* an optimal conditional decision.

Suppose in our course scheduling problem, the full workload requirements have not been revealed to us; specifically, the total number of sessions may be required to be either 10 or 12, and the required length of a practical may be either 1 or 2 hours, but the constraints on the hours per day, and the hours of each session type remain the same. The original problem could then be represented as presented in Figure 21.6, where w_1 and w_3 are known to be 1, but the values of w_2 and t are unknown, but taken from the sets $\{1, 2\}$ and $\{10, 12\}$ respectively. Therefore, there are 4 possible realisations: $\{w_2 = 1, t = 10\}$, $\{w_2 = 1, t = 12\}$, $\{w_2 = 2, t = 10\}$ and $\{w_2 = 2, t = 12\}$. A covering set is shown in Figure 21.7, where the first solution is the most robust solution.

21.3.3 Probability-Based Problems

The next step on from problems with possible realisations is to consider problems where there is a probability distribution over those realisations. Two different formalisms have been proposed with the name *probabilistic CSPs*. The first [25] involves uncertainty over the constraints that appear in the problem, associating a probability with each single constraint, representing the (independent) probability that that constraint is active. The aim is to find an assignment of values to variables which has the highest probability of being a solution to the true problem. For example (Fig. 21.8), suppose we have three possible additional constraints on the practicals in our timetabling problem: the number of practicals must be not less than the number of lectures, with a probability of 0.6; the number of

$$\forall i \sum_{j=1}^{3} w_j x_{ij} > 2 \tag{21.4}$$

$$\forall j \sum_{i=1}^{3} w_j x_{ij} \in \{1, \ldots, 5\} \tag{21.5}$$

$$\sum_{i=1}^{3} \sum_{j=1}^{3} w_j x_{ij} = t \tag{21.6}$$

Figure 21.6: The Uncertain Course Scheduling Problem

practicals on day 3 must be greater than the number of practicals on day 1, with probability 0.5; and the total number of practicals must be no higher than 2, with a probability of 0.2.

There is no assignment with a probability of 1.0 of being a solution; an assignment with maximal probability of being a solution is shown in figure 21.9. This first type of probabilistic CSP could be thought of as the probabilistic equivalent of uncertain CSPs described above, assigning a probability distribution to the values of coefficients in the constraints. Probabilistic CSPs can be represented using the two general soft constraint frameworks *valued* CSP [69] and *semi-ring*[12] CSP described in Chapter 9, "Soft Constraints".

The second type of probabilistic CSPs [26] correspond to mixed CSPs, with the addition of a probability distribution over the possible assignments to the uncontrollable parameters. The aim here is to find a pure decision with maximal probability of being a solution to the full problem. A branch and bound algorithm based on forward checking is described. Again, we can also consider conditional decisions, and an algorithm is given for generating them. Consider now the same problem as described in Figure 21.5, but with two probability distributions over the values of $x_{13} : \{0 : 0.3, 1 : 0.7\}$ and $x_{21} : \{0 : 0.5, 1 : 0.4, 2 : 0.1\}$. Again, no decision can have a probability of 1.0 of being a solution to the full problem (since the two realisations $\langle x_{13} = 1, x_{21} = 2\rangle$ and $\langle x_{13} = 0, x_{21} = 0\rangle$ cannot be satisfied by the same assignment due to the total sessions constraint; figure 21.10 shows a maximal pure decision, with total probability of 0.93 of being a solution (failing only on the realisation $\langle x_{13} = 1, x_{21} = 2\rangle$).

1-stage stochastic CSPs [76] are similar to (the second) probabilistic CSPs, but with the difference that a problem is defined to be θ-satisfiable if there exists a (pure) decision with a probability higher than θ of being a solution. The complexity of 1-stage stochastic CSPs is shown to be NP^{PP}-complete. Stochastic CSPs in general encompass multiple stages and will be discussed further in Section 21.4.3.

21.4 Problems that Change

Now we consider problems that are subject to change over time, and where the opportunity exists to respond to each change via a new solution step. The changes may be imposed by a user, an external agent or the environment. Typically, this occurs during the execution of

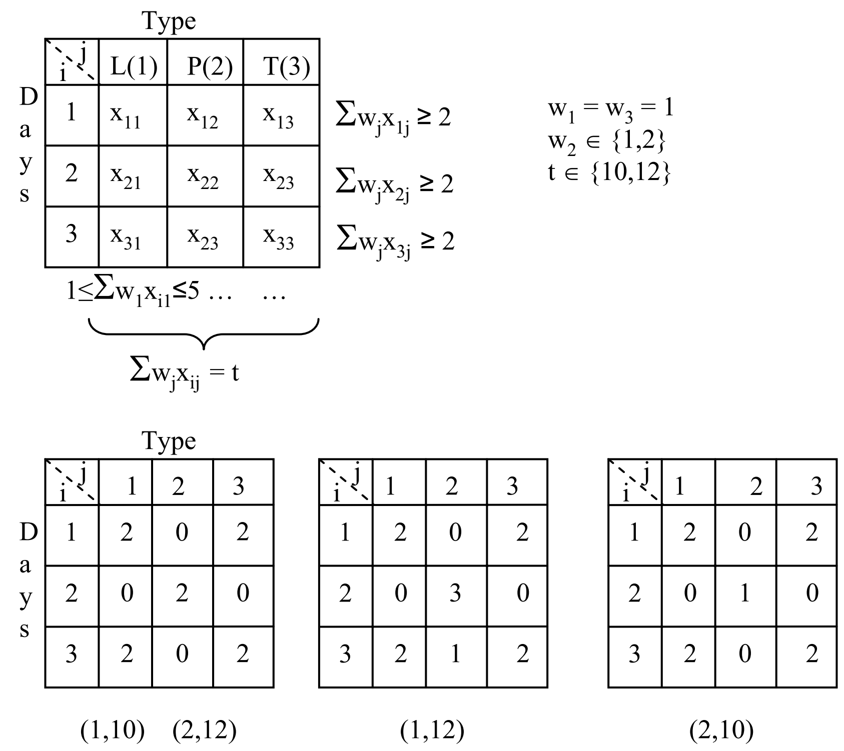


Figure 21.7: *uncertain* CSP: a covering set.

a solution, but in certain cases change may be so rapid that it occurs even as a solution to the original problem is being sought.

Dynamic CSPs ([17], see Figure 21.11) view a changing problem as a sequence of CSPs linked by *restrictions* and *relaxations* (also known as *retractions*), where constraints are respectively added to, and removed from, one problem in the sequence to obtain the next. There are three key concerns in solving dynamic CSPs. The first is to minimise

$$\sum_{i=1}^{3} x_{i2} \geq \sum_{i=1}^{3} x_{i1} \quad (P = 0.6) \tag{21.7}$$

$$x_{32} > x_{12} \quad (P = 0.5) \tag{21.8}$$

$$\sum_{i=1}^{3} x_{i2} \leq 2 \quad (P = 0.2) \tag{21.9}$$

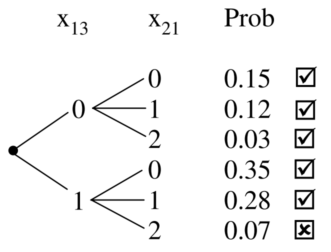
Figure 21.8: The Probabilistic Course Scheduling Problem

		Type		
	i \ j	1	2	3
Days	1	1	0	2
	2	1	1	1
	3	0	2	2

Figure 21.9: *probabilistic* CSP$_1$: a maximal solution.

the need for change, and thus to find *robust*[1] solutions that are likely to remain solutions even after the change has occurred, or to need only minor 'repairs'. The second is to minimise the cost of change, if a change to the solution is required. Hence, we seek *stable* solutions following a change. This is a significant concern in many applications and can stem, for example, from the cost of retooling or simply from the inconvenience to end users. The third is to minimise the reaction time, obtaining a new solution as quickly as possible. The three concerns are often opposed to each other, and thus the particular solution technique implemented will depend on the application. We consider three cases, based on the requirements of the problem and on the knowledge we have of the future changes. Sub-section 21.4.1 assumes no knowledge of the future, and attempts to re-use aspects of the old solution when computing the new solutions. Subsection 21.4.2 also assumes no fore-knowledge of the changes, but attempts to re-use some of the previous reasoning process when generating a new solution. Finally, sub-section 21.4.3 considers problems where the modeller has some uncertain knowledge of the future changes, and examines techniques which are robust to those likely changes. Typically, this involves problems which grow over time, or where the problem structure is gradually revealed.

[1]The term 'flexibility' is also used to describe robustness. For consistency, we use 'robust' throughout.



		Type		
	i \ j	1	2	3
Days	1	2	0	x_{13}
	2	x_{21}	1	1
	3	0	2	2

Figure 21.10: *probabilistic* CSP$_2$: a maximal solution.

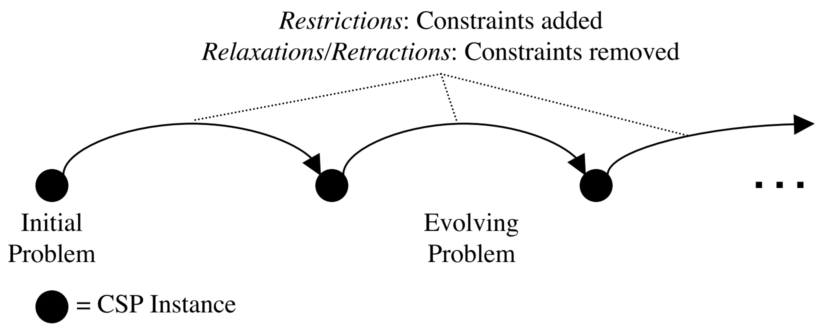

Restrictions: Constraints added
Relaxations/Retractions: Constraints removed
Initial Problem
Evolving Problem
. . .
● = CSP Instance

Figure 21.11: Dynamically Changing Problem Represented as a Sequence of Static CSPs.

21.4.1 Pure Reaction

We will begin by assuming no knowledge of how the problem is likely to change. Naively, each new problem can be solved from scratch. However, efficient solvers exploit the past history of problems and solutions to guide them in solving the new problem, while attempting to minimise the cost of changeover. *Local Repair* methods maintain all assignments from the solution to the previous problem to use as a starting point. The initial assignment is then progressively modified until an acceptable solution to the current problem is obtained.

Minton *et al* [52] describe a local repair method that searches through the space of possible repairs. This search is guided by the *min-conflicts* heuristic that seeks to minimise the number of unsatisfied constraints after each step. The heuristic repair method can be used naturally in a non-systematic (hill-climbing) or systematic (backtracking) search strategy. In the following example, we will illustrate systematic heuristic repair. Reconsider the solution to the Course Scheduling Problem given in Figure 21.2. This solution, although satisfying the constraints given in Figure 21.1, does have a very busy final day. Therefore, the next time the course is run, a new constraint is added that places a maximum on the number of sessions per day. Figure 21.12 presents this variant of the problem, which we will call the *Balanced Course Scheduling Problem*.

Heuristic repair performs a standard backtracking search, with a value ordering heuristic that prefers the assignment that conflicts least with the values assigned by the solution to the previous problem to future variables. Consider solving the Balanced Course Scheduling Problem having obtained the solution to the original Course Scheduling Problem given in Figure 21.2. We use a variable ordering scheme that assigns lecture, then practical then tutorial variables in ascending day order. We also assume that ties are broken by preferring an assignment that matches the previous solution. The current assignments to x_{11} and x_{21} do not conflict with any of the future variables, and so are left unchanged. Consider now the assignment of x_{31}. This variable cannot be assigned 4 or 5, since this would violate constraint (21.2). The remaining values all conflict with the values assigned by the previous solution to x_{32} and x_{33} and constraint (21.10). Since the value 0 is closest to satisfying

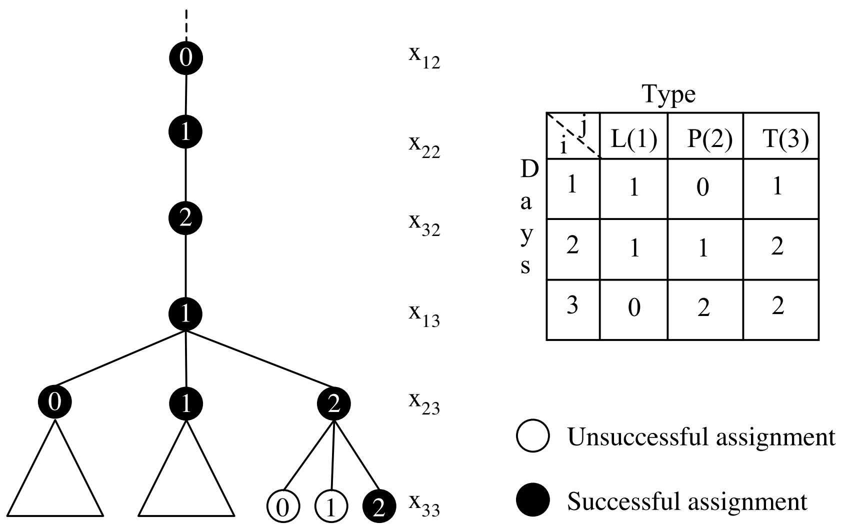
$$\begin{aligned}
\forall i \ \sum_{j=1}^{3} x_{ij} &\geq 2 && \textit{sessions per day} \\
\forall j \ \sum_{i=1}^{3} x_{ij} &\in \{1, 2, \ldots, 5\} && \textit{no. of session type} \\
\sum_{i=1}^{3}\sum_{j=1}^{3} x_{ij} &\in \{10, 11, 12\} && \textit{total sessions} \\
\forall i \ \sum_{j=1}^{3} x_{ij} &\leq 4 && \textit{max sessions per day} \qquad (21.10)
\end{aligned}$$

Figure 21.12: The Balanced Course Scheduling Problem

the constraint[2], it is assigned to x_{31}. The search proceeds in this manner as presented in Figure 21.13.

The *Local Changes* algorithm [73] is also a local repair method, but it uses a more sophisticated search strategy than Minton *et al*'s heuristic repair to focus on resolving the conflicts in a particular sub-problem. Local Changes partitions the variable set X

[2]As noted in [52], for non-binary constraints the measure of conflict depends on the nature of the constraint itself.



i \ j	L(1)	P(2)	T(3)
1	1	0	1
2	1	1	2
3	0	2	2

○ Unsuccessful assignment

● Successful assignment

Figure 21.13: Partial Search Tree for Balanced Course Scheduling Problem using Min-conflicts Heuristic

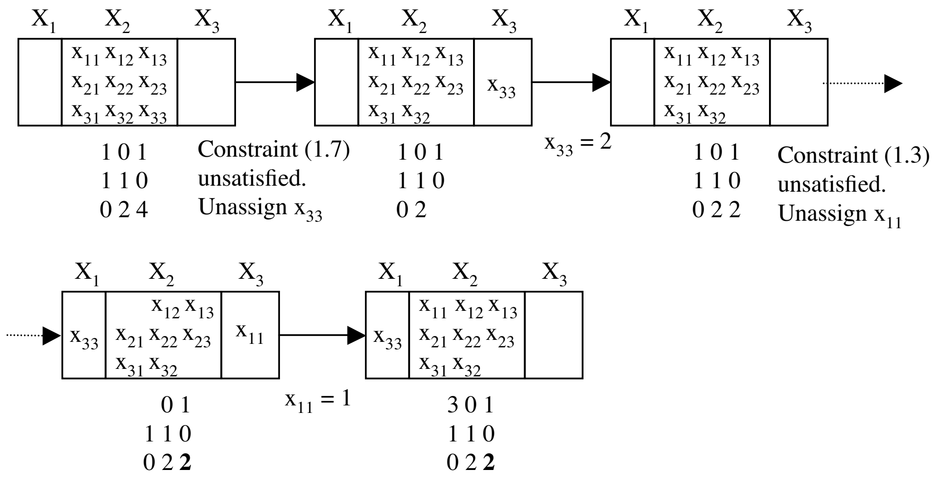


Figure 21.14: Solving the Balanced Course Scheduling Problem using Local Changes

into three subsets, X_1, X_2 and X_3: variables in X_1 have fixed assignments (this is to ensure termination, as will be shown); variables in X_2 have assignments, but which may be modified; variables in X_3 are unassigned. When solving a new problem in a dynamic sequence, all variables are in X_2, with assignments taken from the solution to the previous problem in the sequence. Hence, when solving the Balanced Course Scheduling Problem, search begins with X_2 containing all nine x_{ij} variables, assigned as shown in Figure 21.2.

If this assignment satisfies all constraints, then there is already a solution to the current problem and Local Changes terminates. Otherwise, it unassigns at least one variable for each unsatisfied constraint (placing each in X_3) and attempts to repair their assignments in order to resolve the conflict. Returning to the solution of the Balanced Course Scheduling Problem, as depicted in Figure 21.14, the only constraint that is unsatisfied is the instance of constraint (21.10) concerning day 3. The choice of which of the variables constrained by constraint (21.10) is heuristic. Assume x_{33} is chosen, unassigned and therefore moved into X_3. Local Changes now recurses over X_3, re-assigning the variables to repair the conflicts.

In the example, X_3 contains only x_{33}, which is selected for re-assignment. We assume a reasonably informed value heuristic, assigning $x_{33} = 2$. However, this assignment does not satisfy Constraint (21.3). At this point, Local Changes fixes the assignment of x_{33}, moving it into X_1 and attempts to repair the problem with respect to this choice. The fixing step is to avoid an endless cycle of repairs. If the problem cannot be solved with respect to this assignment, Local Changes will backtrack over it and try another assignment. In the example, x_{11} is re-assigned to 3, producing a solution to the problem. We have demonstrated the operation of Local Changes on a standard dynamic CSP. The algorithm has also been extended to work with fuzzy dynamic CSPs [51] (see Section 21.3.1).

The use of a local repair technique promotes stability by tending to find a solution to the new problem that is close to the solution of the previous problem, as demonstrated by the Min-conflicts and Local Changes examples above. There is no guarantee, however, that the solution will be optimally stable. The alternative is to make stability an explicit criterion when solving each problem in a dynamic sequence, and insist that each new solution is optimally stable. The algorithm RB-AC [66] follows exactly this approach, starting with the solution to the previous problem in the sequence and iteratively testing whether reassigning one variable, two variables, three variables, and so on, is sufficient to solve the current problem. Petcu and Faltings [59] also search explicitly for stable solutions, but do not restrict stability to mean simply the number of assignments in common. Instead, special stability constraints are added that must be satisfied in order for the solution to be stable. Similarly, El Sakkout and Wallace [23] define linear *minimal perturbation* functions for dynamic scheduling problems. Following a change the minimal perturbation function is defined with respect to the solution to the previous problem and used as an objective for the new problem. Bartak *et al* [2] extend this formulation to support over-constrained problems.

21.4.2 Preparing to React by Recording Information

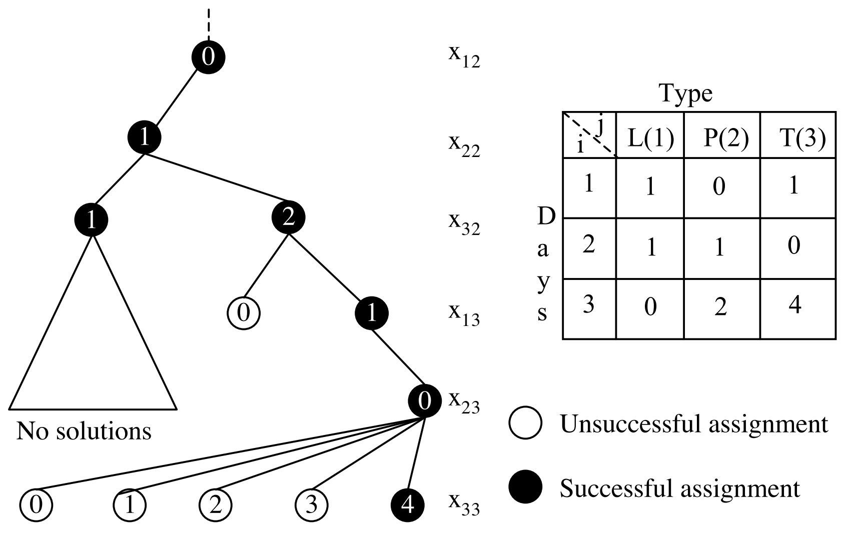
While maintaining our assumption that we have no information about how the problem is likely to change, it is still possible to prepare for these changes by recording information during the search for a solution that is likely to be useful when solving the changed problem, under the reasonable assumption that the latest problem in a dynamic sequence will retain some structure in common with the previous problems.

For each problem in a dynamic sequence, the *oracles* approach [71] records the path taken to the solution. For every new problem in the sequence, search begins from scratch, but these oracles are used to guide the search and prune the search space. Consider first constraint restriction. Figure 21.15 presents a partial search tree for the solution given in Figure 21.2 to the Course Scheduling Problem.

Having solved the Course Scheduling Problem, to solve the Balanced Course Scheduling Problem using the oracles approach, search begins from scratch, using the solution path from Figure 21.15 as the oracle. The search branch down to x_{22} is identical to that explored in finding the previous solution. However, when considering x_{32}, it is possible to prune the sub-tree rooted with $x_{32} = 1$ without exploring it (see Figure 21.15): since there was no solution in this sub-tree for the less-constrained previous problem there cannot be a solution in the sub-tree following constraint restriction. Search continues in this way, as presented in Figure 21.15 following the oracle and pruning fruitless sub-trees until the constraints added cause failure, at which point the search defaults to chronological backtracking while recording a new oracle for future use.

When both restriction and relaxation/retraction are allowed, to retain soundness the oracle chosen must be associated with a previously-solved problem that is less constrained (i.e. contains a subset of the constraints) than the current problem. Van Hentenryck and Provost [71] show how to select an oracle that prunes maximally without sacrificing soundness. Having identified such an oracle, it is used exactly as in the foregoing example.

A popular and powerful approach to preparing for change is to record *explanations*. Jussien [42] defines explanations informally as ”subsets of constraints justifying solver events”. Usually, the solver events are constraint additions, either unary (value removals) or higher arity. Crucially, explanations support change to the problem structure *during*



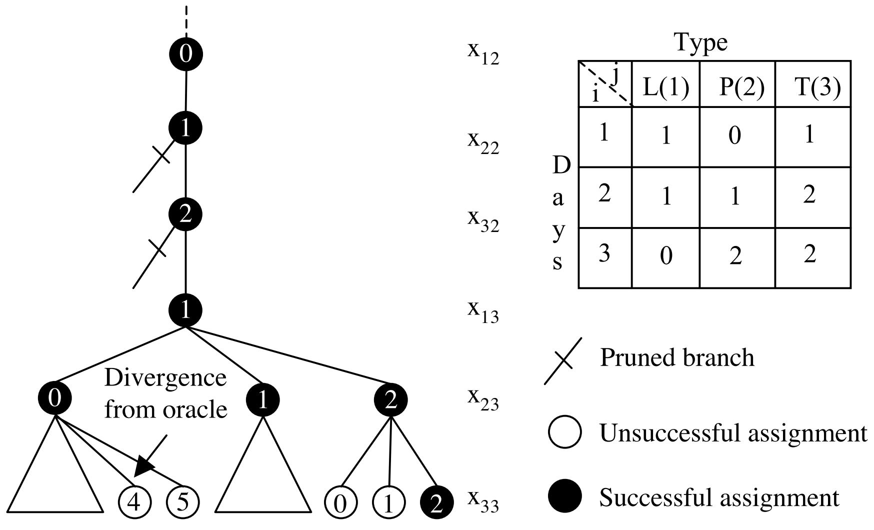
i \ j	L(1)	P(2)	T(3)
1	1	0	1
2	1	1	0
3	0	2	4

Figure 21.15: Partial Search Tree for Course Scheduling Problem

search, as well after a solution has been found and is being executed. Note, however, that supporting changes during systematic search requires a more sophisticated search strategy than simple chronological backtracking, such as Dynamic Backtracking [33, 44] or the Local Changes algorithm discussed in the previous sub-section.

A significant amount of attention in the literature has been devoted to employing explanations in maintaining arc consistency (the reader is directed to Chapter 3 for an explanation of arc consistency) in the face of changes to the problem. Specifically, the problem is assumed to be in an arc consistent state, a change to the problem structure occurs and the goal is to restore arc consistency. Since it is common practice to maintain arc consistency during search, following a change it is natural to restore arc consistency before proceeding. We might also wish to maintain the problem in an arc consistent state, rather than solve it immediately. For instance, Debruyne [15] describes how a bioinformatics problem is configured through a process of interaction with a biologist. The biologist adds or removes constraints from the problem until the current problem is acceptable to him/her. The problem is sufficiently difficult to make solving it following each change impractical, but if enforcing arc consistency does not show that the current problem is unsolvable then this is a good indicator that the problem has solutions. Boyd and Bowen also use explanations to support a similar interactive process [13].

As has been pointed out by many authors, constraint restriction alone is simple to deal with in this setting: a standard arc consistency algorithm can be run as normal following the addition of new constraints. Constraint relaxation/retraction is, however, more difficult to support. This is because value removals resulting from enforcing arc consistency before constraint retraction may no longer be valid. Hence, following retraction, some values typically must be reinstated. Explanations are used to support the identification of these values.



		Type		
	i \ j	L(1)	P(2)	T(3)
Days	1	1	0	1
	2	1	1	2
	3	0	2	2

Figure 21.16: Partial Search Tree for Balanced Course Scheduling Problem using Oracles

One common explanation scheme for this purpose, as embodied by the algorithms DnAC-4 [9], DnGAC4 [10], DnAC-6 [15] as well as the work of Prosser *et al* [64], is based on recording *justifications* for value removals similar to those used in truth maintenance schemes [21]. This is simply the constraint c whose revision caused the value v to be removed from the domain of some variable x. If c is subsequently retracted, v is tentatively restored to x's domain (tentatively because there may be alternative justifications for its removal). Of course the reinstatement of v calls into question all values v' removed from the domains of other variables, specifically where the removal is justified by a constraint involving x. If a constraint check reveals that a v' is supported by v, it is also tentatively restored. This process propagates through the network, restoring values as appropriate. The final step is to run a modified arc consistency algorithm, which removes all tentatively restored values for which it can find an alternative justification.

One variant of this scheme, appearing in the AC|DC algorithm [56] and its descendants [55, 70, 1] saves space by extracting explanations directly from the constraint graph. Another, such as [16], strengthens the justifications recorded to the set of original problem constraints that imply a value removal. The tradeoff is the time and space required to record explanations versus the time required to react to a change in the problem. Although maintaining arc consistency was the original focus of much of this research, explanations have also been used to support the re-use of nogoods discovered during search [68], and have been generalised to arbitrary constraint propagators in, for example, the PaLM system [16] and Constraint Logic Programming [32].

To illustrate, we present a simple example of the utility of explanations. Returning to the original Course Scheduling problem from Figure 21.1, consider that the different session types are indistinguishable — in any (non-)solution, the assignments to one column of variables representing a session can be exchanged with another to produce a (non-)solution.

This is a *symmetry* (see Chapter 10)[3], which can be exploited by imposing an ordering on the session types, for instance by insisting that the sum of the columns is non-decreasing:

$$\sum_{i=1}^{3} x_{i1} \leq \sum_{i=1}^{3} x_{i2} \leq \sum_{i=1}^{3} x_{i3} \qquad \textit{Order Constraints on Session Types} \qquad (21.11)$$

From the total sessions constraint (21.3), one can reason that the session type with the smallest number of assigned sessions can have at most 4 sessions assigned. The ordering constraints (21.11) allow us to identify this session type as the lectures. Hence we can add the implied constraint:

$$\sum_{i=1}^{3} x_{i1} \leq 4 \qquad \textit{Lectures — Revised Maximum} \qquad (21.12)$$

The explanation for constraint (21.12) is the pair of constraints (21.3) and (21.11). Consider now the transition to the Balanced Course Scheduling Problem. Assuming that the ordering constraints to exploit symmetry are retained, the explanation for constraint (21.12), and therefore the constraint itself, remains valid. The saving made is that the cost of deriving the implied constraint is incurred only once, but the benefit, in terms of reducing search following changes to the problem, remains for as long as its explanation is valid.

21.4.3 Predicting Changes

In many real-world problem domains, we have some uncertain knowledge of what the changes might be. For example, in a scheduling problem, we may know the characteristics of all jobs set for production, even if we don't know when the work can begin; a dispatch service may have extensive histories of previous work requests and thus can predict the pattern of future request; or in a manufacturing environment, we may have knowledge of the reliability of a process, and thus can compute the probability of errors. In all of these cases, we can improve our initial solutions by reasoning about the likely changes. In general, we wish to produce *robust* solutions that, when change occurs, are likely to remain solutions or can be modified at little cost.

In *recurrent* CSP [75], changes to problems are assumed to be temporary and recurring — for example, the occasional temporary loss of a resource due to reliability problems. The authors assume that they have no *a priori* knowledge of the changes, and thus must learn the distribution by monitoring changes while solutions are being executed. They propose a min-conflicts [52] repair-based method, to recover solutions when the changes happen, and as they learn the distribution of the changes, they penalise solutions which use values that are frequently lost. In their *supersolutions* framework [37], Hebrard *et al.* address a similar problem, in that values may be unavailable when the solution is executed. Their aim is to find initial solutions that are robust to this loss, or that can be repaired with a small number of changes. They define the concept of an (a, b)-*super solution*, which is a solution to the original problem which, if any a value assignments are lost, can be repaired by reassigning the relevant variables plus another b variables. In particular, a $(1, 0)$-super

[3]The reader will have noticed that the days are also indistinguishable, but we focus on the session types for simplicity.

$$\forall i\ D_i = \{y_{ij} : j = 1 \dots 6\}, count(D_i, \phi) \leq 4 \tag{21.13}$$
$$S = \{y_{ij} : i = 1 \dots 3, j = 1 \dots 6\} \tag{21.14}$$
$$count(S, L) \in \{1, 2, \dots, 5\} \tag{21.15}$$
$$count(S, P) \in \{1, 2, \dots, 5\} \tag{21.16}$$
$$count(S, T) \in \{1, 2, \dots, 5\} \tag{21.17}$$
$$count(S, \phi) \in \{6, 7, 8\} \tag{21.18}$$

Figure 21.17: The Extended Course Scheduling Problem

solution is essentially robust to the loss of any single value — for each variable, there is a backup value which could be assigned without violating any of the constraints.

As an example, consider a more detailed version of the course scheduling problem. We now assume there are six possible time slots each day (giving 18 variables y_{ij}, where $i \in \{1, 2, 3\}$ and $j \in \{1, \dots, 6\}$), which we may fill will a lecture (L), a practical (P) or a tutorial (T), or leave empty (ϕ). The new model is given in Figure 21.17, where we assume a constraint $count(S, v)$, which counts the number of times a variable from the set S takes the value v.

We now assume that after we construct and advertise the timetable, we may be told that certain time slots cannot be filled with sessions of a given type (for example, because of room changes elsewhere). Can we find a $(1, 0)$-supersolution — that is, a solution that can be adapted by reassigning only the affected variable? Figure 21.18a shows one such supersolution — any class (L,P or T) can be replaced by another class, and any empty slot can be filled by a class. Figure 21.18b shows a solution that is not a $(1, 0)$-supersolution, since if we lose the value T from y_{14}, then we cannot find another satisfying solution reassigning only that time slot (since we cannot satisfy the constraint on the number of tutorials).

Periods

Days

i \ j	1	2	3	4	5	6
1	L	P	P	T	0	0
2	L	P	T	0	0	0
3	L	L	T	0	0	0

(a)

i \ j	1	2	3	4	5	6
1	L	P	P	T	0	0
2	L	P	P	0	0	0
3	L	L	P	0	0	0

(b)

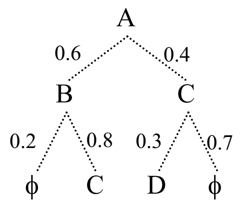
Figure 21.18: *supersolutions*: *(a)* $(1, 0)$-supersolution; *(b)* not a $(1, 0)$-supersolution.

Finding an (a, b)-super solution is shown to be NP-complete for any fixed a. The authors develop a MAC algorithm for finding $(1, 0)$-super solutions, and extend it to a branch-and-bound algorithm for finding the most robust solution when a $(1, 0)$-super solution does not exist (the most robust solution is defined to be one in which a maximal number of variables can be repaired without violating any constraints). This work has been extended [38] to consider $(1, b)$-super solutions, with the ability to place restrictions on the repairs that are considered — for example, to model scheduling problems, where values represent the times at which activities start, the repairs are restricted to using higher values representing later times, so that the repair can be carried out when the break arises during execution. The supersolutions concept has then further extended [39] to include the probability of a value assignment being lost, and the cost of making the repair: specifically, a (α, β)-*weighted supersolution* is one in which any set of value assignments with a total probability greater then α of being lost can be repaired by changing any variables at a total cost of less then β. Weighted supersolutions have been defined to model combinatorial auctions, where each winning bid has a probability of being withdrawn.

Stochastic CSPs [76] (introduced in subsection 21.3.3) allow us to model problems with multiple phases: first the solver must assign a set of variables, then the environment reveals the values of a set of parameters, the solver must then assign another set, and so on. The values of the parameters are assumed to be described by probability distributions. The solution to a multi-stage stochastic CSP is then a tree, in which the assignment of values to the later decision variables are conditional on the previous decisions and the revealed values of the parameters. This allows us to model, for example, production planning, in which the volume to be manufactured in the 2nd quarter depends on the volume manufactured in the 1st quarter, on the realised demand for the 1st quarter, and on the uncertain demand in the future. In the general case, multi-stage stochastic CSPs are $PSPACE$-complete. This work is then extended to use scenario-based semantics [50], and allows chance constraints, which must be satisfied over a proportion of the scenarios. The framework has been implemented as *Stochastic OPL*, in which multiple futures are represented as separate scenarios which are then reformulated as a single larger CSP.

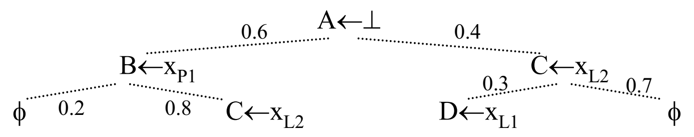
Branching CSP [30] also considers multiple phases, but models problems which grow by the uncertain addition of variables and their associated constraints — for example, online scheduling, where new tasks arrive as the existing tasks are being executed. The model of future arrivals is a probabilistic tree, in which the arrival of any variable is conditional on the preceding arrival sequence. Each variable that arrives may be accepted and assigned a value which does not violate any constraint over the arrived variables, or rejected and assigned no value; a specified utility is gained for each variable that is accepted. The aim is then to assign values to nodes in the tree, such that no constraint is violated and expected utility is maximised. The solution is thus a policy, specifying actions for each possible arrival sequence. Branching CSP has similarities to Markov Decision Problems [65], since the arrivals tree is essentially a finite horizon markov process; however, it is complicated by the fact that choice available at each node is constrained by the previous choices, and formulating the problem as an MDP may require exponentially many states. The Branching CSP algorithms use backtrack search and constraint propagation to reduce this combinatorial explosion [29].

Consider now a special case of the course scheduling problem, in which the resource allocator must decide on initial room requests, but should also cater for new timetabling requests. For simplicity, we consider a simpler problem (Figure 21.19), with one room suitable for lectures, and one for practicals, and three time periods. We assume one initial request: (A) a one hour lecture to be followed by a later one-hour practical. There are also three other requests that we might receive: (B) a two-hour practical, (C) a one-hour lecture followed immediately by a one-hour practical, or (D) another single one-hour lecture. Each requests must be given a time slot immediately or rejected. Each request generates revenue, if it is allocated a time slot; rejected requests generate no revenue. The constraints and the probability tree are shown in the figure. Our immediate task is to decide whether to accept or reject requests A and B, and to allocate times, but ultimately we want a policy for the tree which maximises expected revenue. One example policy is also shown in the figure, which maximises expected revenue by immediately rejecting the unprofitable A, allowing the more profitable B or C, or both, to be accommodated if they arrive.



Constraints
(A): $\{(x_{Li},x_{Pj}): i < j\}$
(B): $\{x_{P1},x_{P2}\}$
(C): $\{x_{L1},x_{L2}\}$
(D): $\{x_{L1},x_{L2},x_{L3}\}$
(A,B): $\{((x_{L1},x_{P3}),x_{P1}),((x_{L2},x_{P3}),x_{P1})\}$
(A,C): $\{((x_{L1},x_{P2}),x_{L2}),((x_{L2},x_{P3}),x_{L1})\}$
(A,D): $\{((x_{Li},x_{Pj}),x_{Lk}): k \neq i\}$
(B,C): $\{(x_{P1},x_{L2})\}$
(C,D): $\{(x_{Li},x_{Lj}): i \neq j\}$

Request	A	B	C	D
Revenue	3	6	9	3



Periods

	1	2	3
L	ϕ	ϕ	ϕ
P	B_1	B_2	ϕ

Pr=0.12, R=6

Periods

	1	2	3
L	ϕ	C_1	ϕ
P	B_1	B_2	C_2

Pr=0.48, R=15

Periods

	1	2	3
L	D	C_1	ϕ
P	ϕ	ϕ	C_2

Pr=0.12, R=12

Periods

	1	2	3
L	ϕ	C_1	ϕ
P	ϕ	ϕ	C_2

Pr=0.28, R=9

Expected revenue = 11.88

Figure 21.19: A branching CSP problem and solution

Bent and van Hentenryck [6, 7] also consider problems which grow over time by the addition of tasks. However, rather than have an explicit probability distribution over the future states, they assume that they have a black-box generator which can generate samples of the future. At each stage in the process, they generate a number of samples, and use the results of optimisation on the samples to make a decision for the current time step. They consider a number of approaches, including: *expectation*, in which each possible decision is evaluated over all samples, and the one with the highest expected value is selected; *con-*

sensus, in which each sample is solved to optimality, and from the solutions the immediate decision which occurs most often is selected; and *regret*, in which each sample is again solved to optimality, and then the possible decisions are evaluated with respect to how much of the objective value would be lost compared to the other decisions. The expectation method produces the best results, but is infeasible for real problems because of the number of optimisations required. The regret method approaches the quality of expectation when there is time to optimise, but is similar to consensus when only a small number of samples are possible, and thus is particularly effective in real-time situations or where the underlying optimisation problem is hard. In common with the approaches that use explicit probability distributions, there is a question as to where the underlying distribution for the black-box generator comes from; the authors have proposed an online learning method [8], which gradually constructs the distribution as it receives requests. [5] also considers problems that grow, examining a number of different approaches to generating robust initial solutions and regular updates

The most significant application area for constraint problems that change is scheduling. Many practical scheduling problems can be expressed as Simple Temporal Problems [20], in which constraints specify single intervals between two time points, and solved in polynomial time. [74] considers an extension in which the durations of some tasks are uncertain, and hence some timepoints are decision variables, while others are uncontrollable (using the same terminology as for mixed CSP [27]). The aim is then to find a policy for executing tasks: problems are defined to be *strongly controllable* if a single decision (i.e. an assignment of a value to each decision variable) will produce an executable schedule regardless of the eventual values of the uncontrollable timepoints; and *weakly controllable* if there exists a decision for each possible realisation of the timepoints. The work was further extended [54] to include *dynamically controllable* problems, for which there exists an online policy: the values assigned to the decision variables need depend only on the observed timepoints in order to get an executable schedule. Checking whether a problem is strongly or dynamically controllable is in P, but weak controllability is in $co-NP$. This work has recently been extended to include soft temporal constraints [78], and it is shown that this does not increase the complexity class: in particular, a polynomial algorithm is presented for generating online execution algorithms that optimise over the soft constraints.

Uncertainty in the duration of tasks is a significant issue in more general scheduling problems. [14] examines the introduction of slack time to handle such uncertainty in job-shop problems. They consider three variations: adding extra time to the duration of every task, modifying the constraints to ensure that slack time exists between tasks, and modifying the constraints dependent on the location of the task in the problem. For a given constraint $Y_{st} \geq X_{st} + dur(x)$, the first would change the value of $dur(x)$ to $dur(x) + \sigma(x)$, while the latter two would change the constraint by adding the term $slack(x)$ to the right hand side. The resultant problem can then be solved using existing scheduling algorithms. Experimental evidence shows that the latter two consistently outperform a simple right-shift reactive solution in terms of tardiness, while the former is significantly poorer, but can give better predictions of execution time in problems with high levels of uncertainty. More recent work [3, 4] considers the problem of producing schedules with a given probability of being executed inside a time limit, and with good probabilistic makespans. The authors develop branch and bound algorithms with Monte Carlo simulation at each node, and heuristic algorithms which generate deterministic problems from the means and variances of the task durations. The heuristic algorithms are shown to scale well with larger problems.

For project scheduling problems, Policella *et al* [61] consider notions of robustness based on initial solutions that are partial orders of tasks. They assume that some pairs of tasks have minimum separation constraints, and that each task occupies a known amount of resource. They consider dynamic changes to the problem in the form of partial resource unavailability, or changes in task duration. Their aim is to produce a partial ordering of the tasks such that any allocation of start times consistent with it also satisfies the time and resource constraints. A partial order is then deemed to be robust if it can absorb changes to the problem details during execution — that is, start times can still be assigned with violating the partial order or the problem constraints. Their approach is first to generate a single schedule with fixed start times, and then to "robustify" it by generating a partial order from it. Previous research has shown that this approach can generate more robust schedules than starting with a least commitment approach [62]. The partial orders are based on chains of precedence constraints for individual units of the resource, and greater robustness is obtained by generating independent chains.

Finally, we note some recent research integrating constraint programming techniques with belief networks, for reasoning about a combination of probabilistic and deterministic information. Belief networks have been studied in AI for many years, and represent the probabilistic dependencies between random variables. They can be used to find the most probable value of a variable, given a set of observations of other variables, and can be used to update beliefs as observations are made incrementally. Constraints can be integrated into the networks by representing them implicitly as conditional probability tables on boolean random variables [58], mapping valid combinations to *true* with probability 1.0, and invalid combinations to *false*. However, this loses the benefits of constraint-based search and propagation. [18] instead represent the constraints explicitly, and show how variable elimination methods can be significantly faster on such representations for computing the probability that a given tuple is a solution. That approach, however, requires large amounts of space. Therefore [19] instead develop search algorithms, which combine constraint propagation with search over AND/OR graphs, requiring only linear space.

21.5 Pseudo-dynamic Formalisms

In this section we describe extensions to classical CSP that, while closely related to dynamic CSPs by name or definition, have important differences that we should be careful to recognise.

We begin by emphasising the difference between dynamic CSPs and what are now known [67] as *conditional* CSPs [53][4]. In a conditional CSP, the whole problem is known statically, but parts of it are made active or inactive depending on the assignments of certain variables. For example, in configuring a car it is only necessary to decide the details of a sunroof if the decision has been made that a sunroof is to be fitted. Conditional CSPs are a natural way to model both configuration [53] (see Chapter 24) and planning problems [45] (see Chapter 22).

[4]The potential for confusion stems from the fact that this work was originally presented with the title 'Dynamic Constraint Satisfaction Problems', where 'dynamic' is refers to the fact that the structure of the problem changed based on decisions made during search

Open Constraint Satisfaction Problems (OCSPs [24]) assume a distributed environment and an *open-world* setting, in which the set of variables and constraints is known statically but the variable domains and tuples allowed by the constraints are incrementally discovered by querying different information sources in a network. This is a natural representation for, for example, many e-commerce problems where suppliers might be queried as necessary as to the specifications and possible configurations of their products. Returning to our running example of course scheduling, one might imagine scheduling a larger course, or multiple courses, taught by several people. In this case, the people involved might be queried to discover acceptable numbers of sessions they were willing to teach and constraints on their timetabling. If the problem remained unsolvable, further queries could be made, and so on.

Open CSP makes the further assumption that information-gathering queries are by far the most expensive individual operation that the solver performs, hence the emphasis is on producing a solution with a minimal number of queries. Faltings and Macho-Gonzalez show that, since domains and allowed tuples increase monotonically with each new query, it is unnecessary to know the entire problem structure in order to solve the problem — a solution to a partially-discovered problem is guaranteed to be a solution to the whole problem [24]. They give the *o-search* algorithm to solve OCSPs that improves over the naive approach of simply gathering all domain values and constraint tuples before solving the problem by interleaving querying and solving: new domain values and constraint tuples are sought only if the currently known sub-problem has no solution. The *fo-search* algorithm refines *o-search* by only gathering new domain values and constraint tuples for the portion of the currently-known sub-problem identified as being responsible for the sub-problem having no solution.

OCSP has also been extended to fuzzy CSPs (see Section 21.3.1) and to optimisation problems [24]. In both cases to be able to find an optimal solution without knowing the whole problem structure there is a monotonicity assumption: domain elements and tuples are returned in non-increasing order of membership degree / non-decreasing order of cost. This is a realistic assumption — the participants in the open course scheduling example described above are likely to be happy to respond to queries with their most preferred option first.

Open CSP is very closely related to Interactive CSP (ICSP [47]) in which again domain elements are acquired incrementally in solving a problem. The key difference is that, since at least one of the solution algorithms presented (*Interactive Forward Checking*) acquires *all* domain values for a particular variable that are consistent with respect to the current assignment, there is an implicit assumption that variable domains are finite. OCSP is also closely related to dynamic CSP, since the incremental addition of domain elements and constraint tuples can be viewed as a sequence of problems linked by the relaxation/retraction of unary constraints disallowing the acquired domain elements [49].

21.6 Challenges and Future Trends

As we have seen, there have been many attempts to extend constraint reasoning to handle dynamic and uncertain problems. The attempts all appear to be isolated, with little commonality between them; they define different problem types, and different types of objectives. In particular, it is difficult to compare techniques, since each is typically addressing its own problem variation, and testing them requires generators of the uncertain and dynamic aspects. There is a need for general purpose, parameterisable, problem generators and execution simulators. Such tools should allow the different types of uncertainty and change to be expressed, and should allow the temporal nature of the changes to be described. An initial scheme for a generator for scheduling problems has been proposed [60]. Tools of this sort would be a start on the road to classifying techniques, and identifying which methods are best suited to which problem types. A common library of problems would be useful in itself, to give an indication of the range and frequency of the different problem types in practical applications. For example, CSPLib[5], an otherwise invaluable repository of benchmark constraint problems, contained no problems with explicit uncertainty or dynamism.

A related challenge is to bring all the different frameworks together. There are some foundational approaches, like Dynamic CSP [17], but nothing as yet with a similar coverage to semiring CSP [12] or valued CSP [69] for soft constraints. Can we find a single framework that encompasses all the different features proposed so far? One such framework has recently been proposed [63], and the question remains open as to whether such a framework should have a rich language allowing the direct expression of many different features, or a simpler more restricted language which would require the reformulation of problems.

On an abstract level, there are three main solution techniques: extending the representational power and reasoning methods to represent uncertain and dynamic problems explicitly, and generate their solutions; reformulating problems into large deterministic problems, and generating the solutions using existing techniques; or generating scenarios or samples, and then solving each one using standard deterministic techniques. It is an open question as to where the boundary lies, to allow us to decide which technique should be applied to which class and size of problem. In particular, more tractability results are required for the different formulations.

In general, constraint solving under change and uncertainty is in its infancy. Closer links need to be established with the existing techniques in other areas of artificial intelligence, mathematics and optimisation, including belief networks [40], MDPs [65] and POMDPs [57], queuing theory [35], stochastic processes [41], stochastic programming [11], Monte Carlo methods [28], stochastic satisfiability [48], decision theory [34] and fuzzy logic [46]. See Halpern [36] for an overview of uncertainty reasoning in general.

Finally, the biggest challenge is to integrate dynamic and uncertain reasoning methods with industrial strength constraint programming tools — as has begun to be the case with, for example, the PaLM system [43]. This would allow the approaches discussed in this chapter and future techniques to be put into practice for real-world decision and optimisation problems, without requiring users to write their own search and propagation algorithms. Towards this goal, Fromherz and Conley [31] describe a general constraint solver design to support a dynamic environment. Further progress is likely to be made by integrating principled simulation and sampling techniques first — see for example [50] — since they will allow existing CP tools to be used without modification.

[5]http://www.csplib.org, 29th September, 2005

21.7 Summary

Many real and important problems involve change and uncertainty. Solutions are required that take account of vagueness in the problem description, or that minimise the effect of the uncertainty on the solution. Basic approaches to handling change include rapid reaction through re-specifying the problems and re-solving when the changes occur, preparing to change by maintaining explanations and data structures that will allow the solver to avoid repeating work, or proactively generating solutions that are robust, by explicitly reasoning about the possible changes. A number of different techniques have been developed, and they have demonstrated that constraint programming methods can be extended to handle many different forms of dynamism and uncertainty, and that many exemplar problems can be solved efficiently. Constraint programming toolkits need to be extended with facilities to handle such problems. Further work is required to establish which of the techniques and frameworks are practical candidates, and to integrate this body of research with the many other research fields which deal with change and uncertainty. Finally, for an alternative viewpoint on the material in this chapter, the reader is directed to the survey by Verfaillie and Jussien [72].

Acknowledgments

We thank the anonymous referee for useful comments on a draft of this chapter. Ken Brown's work was supported in part by grants 03/CE3/I405 (SFI Centre for Telecommunications Value-chain Research) and SC/2003/81 (Enterprise Ireland). Ian Miguel is supported by a UK Royal Academy of Engineering/EPRSC Research Fellowship.

Bibliography

[1] R. Bartak and P. Surynek. An improved algorithm for maintaining arc consistency in dynamic constraint satisfaction problems. In *FLAIRS'05: Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference*, pages 161–166. AAAI Press, 2005.

[2] R. Bartak, T. Muller, and H. Rudova. A new approach to modeling and solving minimal perturbation problems. In *Recent Advances in Constraints*, volume 3010, pages 223–249. Springer Lecture Notes in Artificial Intelligence, 2004.

[3] J. C. Beck and N. Wilson. Job shop scheduling with probabilistic durations. In *ECAI'04: Proceedings of the Sixteenth European Conference on Artificial Intelligence*, pages 652–656. IOS Press, 2004.

[4] J. C. Beck and N. Wilson. Proactive algorithms for scheduling with probabilistic durations. In *IJCAI'05: Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 1201–1206. Professional Book Center, 2005.

[5] T. Benoist, E. Bourreau, Y. Caseau, and B. Rottembourg. Towards stochastic constraint programming: A study of online multichoice knapsack with deadlines. In *CP'01: Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming*, volume 2239, pages 61–76. Springer Lecture Notes in Computer Science, 2001.

[6] R. Bent and P. van Hentenryck. The value of consensus in online stochastic scheduling. In *ICAPS'04: Fourteenth International Conference on Automated Planning and Scheduling*, pages 219–226. AAAI Press, 2004.

[7] R. Bent and P. van Hentenryck. Regrets only! online stochastic optimization under time constraints. In *AAAI'04: Proceedings of the Nineteenth National Conference on Artificial Intelligence*, pages 501–506. AAAI Press, 2004.

[8] R. Bent and P. van Hentenryck. Online stochastic optimization without distributions. In *ICAPS'05: Fifteenth International Conference on Automated Planning and Scheduling*, pages 171–180. AAAI Press, 2005.

[9] C. Bessiere. Arc-consistency in dynamic constraint satisfaction problems. In *AAAI'91: Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 221–226. AAAI Press/MIT Press, 1991.

[10] C. Bessiere. Arc-consistency for non-binary dynamic CSPs. In *ECAI'92: Proceedings of the Tenth European Conference on Artificial Intelligence*, pages 23–27. John Wiley and Sons, 1992.

[11] J. R. Birge and F. V. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, 1997.

[12] S. Bistarelli, U. Montanari, and F. Rossi. Constraint solving over semi-rings. In *IJCAI'95: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 624–630. Morgan Kaufmann, 1995.

[13] D.B. Boyd and J. Bowen. Using dependency records to generate design coordination advice in a constraint-based approach to concurrent engineering. *Computers in Industry*, 33(2):191–199, 1997.

[14] A. J. Davenport, C. Gefflot, and J. C. Beck. Slack-based techniques for robust schedules. In *ECP'01: Proceedings of the Sixth European Conference on Planning*, pages 7–18, 2001.

[15] R. Debruyne. Arc-consistency in dynamic CSPs is no more prohibitive. In *ICTAI'96: Proceedings of the Eighth International Conference on Tools with Artificial Intelligence*, pages 299–307. IEEE Computer Society, 1996.

[16] R. Debruyne, G. Ferrand, N. Jussien, W. Lesaint, S. Ouis, and A. Tessier. Correctness of constraint retraction algorithms. In *FLAIRS'03: Proceedings of the Sixteenth International Florida Artificial Intelligence Research Society Conference*, pages 172–176. AAAI Press, 2003.

[17] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *AAAI'88: Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 37–42. AAAI Press/MIT Press, 1988.

[18] R. Dechter and D. Larkin. Hybrid processing of belief and constraints. In *UAI'01: Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 112–119. Morgan Kaufmann, 2001.

[19] R. Dechter and R. Mateescu. Mixtures of deterministic-probabilistic networks and their and/or search space. In *UAI'04: Proceedings of the Twentieth Annual Conference on Uncertainty in Artificial Intelligence*, 2004.

[20] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[21] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12:231–272, 1979.

[22] D. Dubois, H. Fargier, and H. Prade. Possibility theory in constraint satisfaction problems. *Applied Intelligence*, 6:287–309, 1996.

[23] H. El Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.

[24] B. Faltings and S. Macho-Gonzalez. Open constraint programming. *Artificial Intelligence*, 161, 2005.

[25] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probalistic approach. In *ECSQARU'93: Proceedings of the Second European Conference on Symbolic and Qualitative Approaches to Reasoning with Uncertainty*, volume 747, pages 97–104. Springer Lecture Notes in Computer Science, 1995.

[26] H. Fargier, J. Lang, R. Martin-Clouaire, and T. Schiex. A constraint satisfaction framework for decision under uncertainty. In *UAI'95: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 167–174. Morgan Kaufmann, 1995.

[27] H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In *AAAI'96: Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 175–180. AAAI Press/MIT Press, 1996.

[28] G. S. Fishman. *Monte Carlo: Concepts, Algorithms and Applications*. Springer Verlag, 1996.

[29] D. Fowler and K. Brown. Branching constraint satisfaction problems and markov decision problems compared. *Annals of Operations Research*, 118:85–110, 2003.

[30] D. W. Fowler and K. N. Brown. Branching constraint satisfaction problems for solutions robust under likely changes. In *CP2000: Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, volume 1894, pages 500–504. Springer Lecture Notes in Computer Science, 2000.

[31] M. Fromherz and J. Conley. Issues in reactive constraint solving. In *COTIC'97: Proceedings of the Workshop on Concurrent Constraint Programming for Time Critical Applications*, 1997.

[32] Y. Georget, P. Codognet, and F. Rossi. Constraint retraction in CLP(FD): Formal framework and performance results. *Constraints*, 4(1):1–41, 1999.

[33] M. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1: 25–46, 1993.

[34] P. Goodwin and G. Wright. *Decision Analysis for Management Judgment (3e)*. Wiley, 2004.

[35] D. Gross and C. M. Harris. *Fundamentals of Queueing Theory (3e)*. Wiley, 1998.

[36] J. Halpern. *Reasoning about Uncertainty*. MIT Press, 2003.

[37] E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *CPAIOR'04: Proceedings of the First International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, volume 3011, pages 157–172. Springer Lecture Notes in Computer Science, 2004.

[38] E. Hebrard, B. Hnich, and T. Walsh. Robust solutions for constraint satisfaction and optimization. In *ECAI'04: Proceedings of the Sixteenth European Conference on Artificial Intelligence*, pages 186–190. IOS Press, 2004.

[39] A. Holland and B. OŚullivan. Weighted super solutions for constraint programs. In *AAAI'05: Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 378–383. AAAI Press/MIT Press, 2005.

[40] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Springer, 2001.

[41] P. W. Jones and P. Smith. *Stochastic Processes*. Oxford University Press, 2001.

[42] N. Jussien. The versatility of using explanations within constraint programming. Technical Report 03-04-INFO, Ecole des Mines de Nantes, 2003.

[43] N. Jussien and V. Barichard. The PaLM system: Explanation-based constraint programming. In *TRICS'00: Proceedings of the International Workshop on Techniques for Implementing Constraint Programming Systems*, pages 118–133, 2000.

[44] N. Jussien, R. Debruyne, and P. Boizumault. Maintaining arc-consistency within dynamic backtracking. In *CP'2000: Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming*, volume 1894, pages 249–261. Springer Lecture Notes in Computer Science, 2000.

[45] S. Kambhampati. Planning graph as a (dynamic) CSP: Exploiting EBL, DDB and other CSP search techniques in graphlan. *Journal of Artificial Intelligence Research*, 12:1–34, 2000.

[46] G. Klir and Yuan B. *Fuzzy sets and fuzzy logic: theory and applications*. Prentice Hall, 1995.

[47] E. Lamma, P. Mello, M. Milano, R. Cucchiara, M. Gavanelli, and M. Piccardi. Constraint propagation and value acquisition: Why we should do it interactively. In *IJCAI'99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 467–473. Morgan Kaufmann, 1999.

[48] M. Littman, S. Majercik, and T. Pitassi. Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[49] S. Macho-Gonzalez and P. Meseguer. Open, interactive and dynamic CSP. In *Proceedings of the International Workshop on Constraint Solving under Change and Uncertainty*, pages 13–17, 2005.

[50] S. Manander, A. Tarim, and T. Walsh. Scenario-based stochastic constraint programming. In *IJCAI'03: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 257–262. Morgan Kaufmann, 2003.

[51] I. Miguel and Q. Shen. Fuzzy rrDFCSP and planning. *Artificial Intelligence*, 148 (1–2):11–52, 2003.

[52] S. Minton, M.D. Johnston, A.B. Philps, and P. Laird. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[53] S. Mittal and B. Falkenhainer. Dynamic constraint satisfaction problems. In *AAAI'90: Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 25–32. AAAI Press/MIT Press, 1990.

[54] P. Morris, N. Muscettola, and T. Vidal. Dynamic control of plans with temporal uncertainty. In *IJCAI'01: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 494–502. Morgan Kaufmann, 2001.

[55] M. Mouhoub. Arc consistency for dynamic CSPs. In *KES'03: Proceedings of the Seventh International Conference on Knowledge-based Intelligent Information and Engineering Systems*, volume 2773, pages 393–400. Springer Lecture Notes in Computer Science, 2003.

[56] B. Neveu and P. Berlandier. Maintaining arc consistency through constraint retraction. In *ICTAI'94: Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, pages 426–431. IEEE Computer Society, 1994.

[57] L. Pack Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[58] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[59] A. Petcu and B. Faltings. Optimal solution stability in continuous-time optimization. In *DCR'05: Proceedings of the Sixth International Workshop on Distributed Constraint Reasoning*, pages 207–221, 2005.

[60] N. Policella and R. Rasconi. Looking for a common scheduling perturbations benchmark. In *Changes'05: Proceedings of the International Workshop on Constraint Solving under Change and Uncertainty, Sitges*, pages 23–27, 2005.

[61] N. Policella, A. Oddi, S. F. Smith, and A. Cesta. Generating robust partial order schedules. In *CP'04: Proceedings of the Tenth International Conference on the Principles and Practice of Constraint Programming*, volume 3258, pages 406–511. Springer Lecture Notes in Computer Science, 2004.

[62] N. Policella, S. F. Smith, and A. Cesta, A.and Oddi. Generating robust schedules through temporal flexibility. In *ICAPS'04: Fourteenth International Conference on Automated Planning and Scheduling*, pages 209–218. AAAI Press, 2004.

[63] C. Pralet, G. Verfaillie, and T. Schiex. Composite graphical models for reasoning about uncertainties, feasibilities, and utilities. In *Soft'05: Proceedings of the Seventh International Workshop on Preferences and Soft Constraints, Sitges*, pages 104–118, 2005.

[64] P. Prosser, C. Conway, and C. Muller. A constraint maintenance system for the distributed resource allocation problem. *Intelligent Systems Engineering*, 1(1), 1992.

[65] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[66] N. Roos, Y. Ran, and J. van den Herik. Combining local search and constraint propagation to find a minimal change solution for a dynamic CSP. In *AIMSA'00: Proceedings of the Ninth International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, volume 1904, pages 272–282. Springer Lecture Notes in Computer Science, 2000.

[67] M. Sabin and E. Freuder. Detecting and resolving inconsistency and redundancy in conditional constraint satisfaction problems. In *Proceedings of the CP'98 Workshop on Constraint Problem Reformulation*, 1998.

[68] T. Schiex and G. Verfaillie. Nogood recording for static and dynamic constraint satisfaction problems. *International Journal of Artificial Intelligence Tools*, 3(2): 187–207, 1994.

[69] T. Schiex, H. Fargier, and G. Verfaillie. Valued constraint satisfaction problems: Hard and easy problems. In *IJCAI'95: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 631–637. Morgan Kaufmann, 1995.

[70] P. Surynek and R. Bartak. A new algorithm for maintaining arc consistency after constraint retraction. In *CP'04: Proceedings of the Tenth International Conference on Principles and Practice of Constraint Programming*, volume 3258, pages 767–771. Springer Lecture Notes in Computer Science, 2004.

[71] P. van Hentenryck and T. L. Provost. Incremental search in constraint logic programming. *New Generation Computing*, 9:257–275, 1991.

[72] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

[73] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *AAAI'94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 307–312. AAAI Press, 1994.

[74] T. Vidal and H. Fargier. Handling contingency in temporal constraint networks: From consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence*, 11:23–45, 1999.

[75] R. J. Wallace and E. C. Freuder. Stable solutions for dynamic constraint satisfaction problems. In *CP'98: Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, volume 1520, pages 447–461. Springer Lecture Notes in Computer Science, 1998.

[76] T. Walsh. Stochastic constraint programming. In *ECAI'02: Proceedings of the Fifteenth European Conference on Artificial Intelligence*, pages 111–115. IOS Press, 2002.

[77] N. Yorke-Smith and C. Gervet. Certainty closure: A framework for reliable constraint reasoning with uncertainty. In *CP'03: Proceedings of the Ninth International Conference on Principles and Practice of Constraint*, volume 2833, pages 769–783. Springer Lecture Notes in Computer Science, 2003.

[78] N. Yorke-Smith, K. B. Venable, and F. Rossi. Temporal reasoning with preferences and uncertainty. In *IJCAI'03: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, pages 1385–1386. Morgan Kaufmann, 2003.