Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Artificial Intelligence 148 (2003) 11–52

Artificial Intelligence

www.elsevier.com/locate/artint

Fuzzy *rr*DFCSP and planning

Ian Miguel [a], Qiang Shen [b,*]

[a] *Department of Computer Science, University of York, York YO10 5DD, UK*
[b] *School of Informatics, University of Edinburgh, Edinburgh EH8 9LE, UK*

Received 16 July 2002; received in revised form 1 September 2002

Abstract

Constraint satisfaction is a fundamental Artificial Intelligence technique for knowledge representation and inference. However, the formulation of a static constraint satisfaction problem (CSP) with hard, imperative constraints is insufficient to model many real problems. Fuzzy constraint satisfaction provides a more graded viewpoint. Priorities and preferences are placed on individual constraints and aggregated via fuzzy conjunction to obtain a satisfaction degree for a solution to the problem. This paper examines methods for solving an important instance of dynamic flexible constraint satisfaction (DFCSP) combining fuzzy CSP and restriction/relaxation based dynamic CSP: fuzzy *rr*DFCSP. This allows the modelling of complex situations where both the set of constraints may change over time and there is flexibility inherent in the definition of the problem. This paper also presents a means by which classical planning can be extended via fuzzy sets to enable flexible goals and preferences to be placed on the use of planning operators. A range of plans can be produced, trading compromises made versus the length of the plan. The flexible planning operators are close in definition to fuzzy constraints. Hence, through a hierarchical decomposition of the planning graph, the work shows how flexible planning reduces to the solution of a set of fuzzy *rr*DFCSPs.

Keywords: Constraint satisfaction; Dynamic CSP; Flexible CSP; AI planning; Flexible planning

1. Introduction

Constraints are a natural means of knowledge representation in many disparate fields. A constraint often takes the form of an equation or inequality, but in the most abstract sense is a logical relation among several variables expressing a set of admissible value

* Corresponding author.
E-mail addresses: ianm@cs.york.ac.uk (I. Miguel), qiangs@dai.ed.ac.uk (Q. Shen).

0004-3702/$ – see front matter
doi:10.1016/S0004-3702(03)00020-1

combinations. The following are simple examples: the sum of two variables must equal 30; adjacent countries on the map cannot be coloured the same; the helicopter is designed to carry one passenger, but a second can be carried in an emergency; the maths class must be scheduled between 9 and 11am, but it may later be moved to the afternoon.

Constraint satisfaction is the process of identifying a solution to a problem which satisfies all specified constraints. The classical *Constraint Satisfaction Problem* (CSP) [7,18,33] involves a fixed set of problem *variables*, each with an associated *domain* of potential values. A set of *constraints* range over the variables, specifying the allowed combinations of *assignments* of values to variables. To solve a classical CSP, it is necessary to find one or all assignments to all variables such that all constraints are satisfied. A constraint satisfaction solution procedure must find one/all such assignments or prove that no such solution exists.

Despite its simplicity, a constraint-based representation can express real, difficult problems. For example, the problems of interpreting an image, scheduling a collection of tasks or diagnosing a fault in an electrical circuit can all be viewed as instances of the CSP. One area that involves extensive use of constraint satisfaction is that of AI planning [38]. Graphplan [3] reduces classical domain-independent planning [38] to the solution of a CSP. By this method, large efficiency gains were made as compared to previous state-of-the-art planning algorithms.

As classical constraint satisfaction has been applied to more complex real problems it has become increasingly clear that the classical formulation is insufficient. Consider, for instance, the opening example involving the capacity of the helicopter. In reality, if no other solution could be found the helicopter would carry a second passenger to create a compromise solution. Classical constraint satisfaction supports *hard* constraints which are *imperative* (a valid solution must satisfy all of them) and *inflexible* (constraints are either wholly satisfied or wholly violated). In reality problems rarely exhibit this rigidity of structure. It is common for there to be *flexibility* which can be used to overcome over-constrainedness (i.e., a problem with no solution) by indicating where a sensible compromise can be made [22]. Classical CSP has been extended to incorporate different types of 'soft' constraints often found in real problems. One successful example is fuzzy CSP [9]. Rather than enforcing binary satisfaction/dissatisfaction, it provides a more graded viewpoint through a fuzzy set-based representation. Priorities and preferences are placed on individual constraints and aggregated via fuzzy conjunction to obtain a satisfaction degree for each solution.

Another weakness of classical constraint satisfaction is that it cannot efficiently support problems whose structure is subject to change. Once the sets of variables, domains and constraints have been defined, they are fixed for the duration of the solution process. In reality problems may change (e.g., the opening example concerning the scheduling of the maths class) either as a solution is being constructed, or while a constructed solution is in use. If so, the natural approach is to attempt to repair the old solution, disturbing it as little as possible. Classical constraint satisfaction can deal at best only clumsily with this situation, considering the changed problem as an entirely new problem to be solved from scratch. To address these types of problem, *dynamic* constraint satisfaction techniques have been developed [20]. To model problems which change over time, constraints are added to (constraint *restriction*) and removed from (constraint *relaxation*) the current problem

(*rr*DCSP, [8]). Specialised techniques re-use as much of the solution or partial solution obtained for a problem before it changed with respect to the new problem state [25,34,35].

However, current dynamic constraint satisfaction research is founded almost exclusively on classical CSP, unable to take advantage of fuzzy constraints in a dynamic environment, and fuzzy CSP research is limited to static problems. Little has been done to combine dynamic and fuzzy constraint satisfaction to maintain the benefits of both individual approaches to solve more complex problems. The combined approach will form an important instance of dynamic flexible CSP [23] which will be referred to as *rr*DFCSP. In this paper, an extensive empirical analysis is made of the structure and properties of fuzzy *rr*DFCSP.

Furthermore, this paper applies fuzzy *rr*DFCSP to the field of AI Planning. This field is an active and long-established research area with a wide applicability to such tasks as automating data-processing procedures [5], game-playing [32], and large-scale logistics problems [39]. As per classical CSP, classical AI Planning is unable to support flexibility in the problem description. Providing such an ability is a significant step forward for the real-world utility of planning research. An extension to the classical AI Planning problem is presented here to incorporate fuzzy constraints to create a *flexible planning problem* and to show how flexible plan synthesis is reduced to the solution of a hierarchy of fuzzy *rr*DFCSPs. The flexible planner Flexible Graphplan (FGP) is developed as a means to solve such problems and its performance is analysed theoretically and empirically.

The next section provides a more detailed background on classical, fuzzy and dynamic CSP. Section 3 describes fuzzy *rr*DFCSP and solution methods for these problems. An extensive empirical analysis of fuzzy *rr*DFCSP, using several solution algorithms, is made in Section 4. Section 5 describes the flexible planning problem. Section 6 presents the Flexible Graphplan algorithm for solving such problems, which is analysed experimentally in Section 7. Section 8 concludes the paper and points out important future work.

2. Background

A more detailed description of classical, fuzzy and dynamic constraint satisfaction is reviewed here. The Course Scheduling Problem (adapted from [9]), is used to illustrate the utility of each approach and the need for fuzzy dynamic CSP. This problem consists of deciding the number of lecture, exercise and training sessions for a course. In total, there must be 8 sessions. Professor A agrees to give 4 or 5 lectures, Dr B agrees to give 3 or 4 exercise sessions and there must be an additional 1 or 2 training sessions.

2.1. Classical constraint satisfaction

A classical constraint satisfaction problem (CSP) comprises n variables, $X = \{x_1, \ldots, x_n\}$, each with domain D_i, describing its potential values. A variable, x_i, is *assigned* one of the values from D_i. The Course Scheduling problem uses x_1, x_2, x_3 for the number of each session type. A set of constraints, C, ranges over these variables. A constraint $c(x_i, \ldots, x_j) \in C$ specifies a subset of the Cartesian product $D_i \times \cdots \times D_j$, indicating

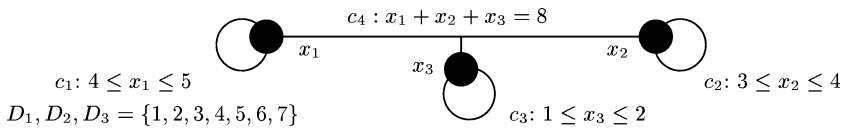

Fig. 1. Course scheduling problem: constraint hyper-graph representation.

variable assignments that are compatible with each other. These constraints are imperative (each one *must* be satisfied) and inflexible (fully satisfied or violated).

A CSP can be represented graphically as a *constraint network* [7], the structure of which is used to guide CSP solution techniques. A common formulation represents variables as nodes and constraints as edges. Fig. 1 shows a constraint hyper-graph representation of the Course Scheduling problem. A *solution* to a classical CSP is a complete assignment to the problem variables satisfying all constraints. A CSP may contain several solutions, and the task for a constraint-based problem-solver is to find one or all of these. A solution to the Course Scheduling problem is $\{x_1 = 4, x_2 = 3, x_3 = 1\}$.

2.2. Fuzzy constraint satisfaction

Classical constraint satisfaction techniques support only *hard* constraints specifying exactly the allowed variable assignment combinations. When an over-constrained problem (i.e., which has no solution) is encountered, some constraints must be *relaxed* (removed or weakened) before a solution to a less-constrained (but still interesting) problem can be found. Without an indication of the relative importance of each constraint it is difficult to do this consistently such that a useful solution will be found. Consider the Inconsistent Course Scheduling problem, a variant of the Course Scheduling problem presented in Fig. 1, where the total number of sessions is reduced to 7. This problem has no solution, how can a useful solution be produced?

Real problems in general are not easily described in such definite terms as classical CSP requires. A simple example is the expression of *preferences* among the set of assignments, either to the whole problem or local to an individual constraint. *Prioritised* constraints are also useful in the case of an over-constrained problem. If constraints with a lower priority are relaxed it is more likely that the eventual solution will be useful. For real-time tasks finding flexible solutions may be the only option. Classical techniques find a perfect solution or no solution at all, presenting a serious problem if time runs out for the problem-solver. A flexible system could return the best solution so far, embodying an *anytime* algorithm, as noted in [11].

Fuzzy Constraint Satisfaction [9] supports prioritised and preference constraints. Both are modelled by a fuzzy relation, R, defined by μ_R, a *membership function* associating an assignment tuple in $D_1 \times \cdots \times D_n$ with a value in a totally ordered *satisfaction scale*, $L = \{l_\bot, l_1, l_2, \ldots, l_\top\}$. A preference constraint, c_i, amongst a set, A, of potential assignments to its constrained variables can be modelled as follows, where $a \in A$:

$$\mu_{R_i}(a) = l_\top \quad \text{if } a \text{ totally satisfies } c_i,$$
$$\mu_{R_i}(a) = l_\bot \quad \text{if } a \text{ totally violates } c_i,$$

$$l_\perp < \mu_{R_i}(a) < l_\top \quad \text{if } a \text{ partially satisfies } c_i.$$

The scale $V = \{v_\perp, v_1, v_2, \ldots, v_\top\}$ (effectively L in reverse) is used to support prioritised constraints, representing *possibility of violation* (priority) of a constraint. A priority degree, $v_j \in V$, is associated with each prioritised constraint, c_j. A constraint with priority $v_\top$ is imperative and a constraint with priority $v_\perp$ is totally irrelevant. The bijection b maps from V to L such that $L = b(V)$. A prioritised constraint c_j with priority v_j is modelled as follows:

$$\mu_{R_j}(a) = l_\top \quad \text{if } a \text{ satisfies } c_j,$$
$$\mu_{R_j}(a) = b(v_j) \quad \text{if } a \text{ violates } c_j.$$

A prioritised-preference constraint is represented by a fuzzy relation R_k, with μ_{R_i} describing the preference component. A constraint c_k, represented by R_k will be satisfied to at least the degree $b(v_k)$ since the priority degree defines a bound on the damage to the solution that is incurred by the violation of this constraint. The constraint can be satisfied above this level by satisfying the preference component to a higher degree.

$$\mu_{R_k}(a) = \max\big(b(v_k), \mu_{R_i}(a)\big).$$

The consistency level of a partial assignment is calculated from the aggregated membership values of the constraints involving the assigned variables as follows, where a is an assignment to $x_1, \ldots, x_k$; $\otimes$ represents the conjunctive combination of fuzzy relations (usually interpreted as the minimum membership value assigned by all relations); and $Vars(R_i)$ is the set of variables constrained by the constraint represented by R_i:

$$cons(a) = \otimes\mu_{R_i}(\Pi_{Vars(R_i)}x).$$

This quantity is an upper bound on the consistency of a complete assignment. It is computed incrementally during search by considering only constraints that involve the current variable, and is commonly used in a branch and bound search to find the best solution [9].

Fig. 2 presents a fuzzy version of the Inconsistent Course Scheduling problem. The fuzzified version allows the identification of the constraints that should be relaxed first and a more precise specification (via preferences) of how each constraint is best satisfied. The wish of Professor A to give about 4 lectures is assigned priority v_3, with preference given

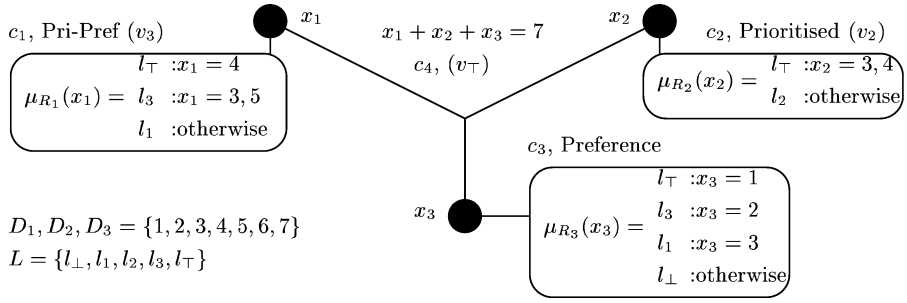

Fig. 2. Fuzzy course scheduling problem.

to values close to 4. Similarly, the wish of Dr B to give 3 or 4 lectures is assigned priority v_2. The fact that there should be about 1 additional training session is represented by a preference constraint. Finally, that there must be 7 sessions in total is a hard constraint and so is assigned priority $v_\top$. As per the Inconsistent Course Scheduling problem, the Fuzzy Course Scheduling problem has no perfect solution. However, a useful compromise solution can still be found, i.e., $\{x_3 = 3, x_2 = 3, x_3 = 1\}$. Since Professor A is relatively happy with giving 3 lectures, the satisfaction degree of c_1 and of the overall solution is l_3.

2.3. Dynamic constraint satisfaction

In the description so far, problems have been assumed to be *static*, precluding any changes to the problem structure after its initial specification. To a certain extent, fuzzy CSP can be seen as adding dynamicity to the problem, which is effectively changed when violated constraints are softened/removed. Many real problems, however, are subject to change caused not by the need to find a compromise, but by the evolution of the problem structure. Techniques for solving *Dynamic Constraint Satisfaction Problems* (DCSPs) address this need.

Although several alternative DCSP formulations exist [20], this paper concentrates on the earliest and most natural choice, as presented in [8]. A dynamic environment is viewed as a sequence of CSPs linked by *restrictions* and *relaxations*, where constraints are added to and removed from the problem respectively. This type of problem will hereafter be referred to as *rr*DCSP. Naively, each individual problem in the sequence may be solved from scratch using static CSP techniques, but this method discards all the work done in solving the previous (probably similar) problem. Efficient *rr*DCSP solvers re-use as much as possible of the effort required to solve previous problems in solving the current problem.

The *oracles* approach [34] searches through previously solved instances for a less constrained version of the current problem. If one is found, the part of the search space before the associated solution is not explored, since no solution exists in it for the more constrained current problem. *Local repair* [25,35] maintains all assignments from the solution to the previous problem to use as a starting point. Individual variable assignments are modified until an acceptable solution is obtained. Constraint recording methods [31,34] infer new constraints from the existing problem definition which disallow inconsistent assignment combinations not directly disallowed by the original constraints. The justifications of inferred constraints are recorded so that they can be used in future problems, where the same justifications hold, to converge on a solution more quickly.

Returning to the Course Scheduling problem (of Fig. 1), consider the effects of changing this problem as shown in Fig. 3. Professor A will now teach only 3 or 4 lectures, but Dr B agrees to give 4 or 5 exercise sessions. The naive reaction is to simply apply a

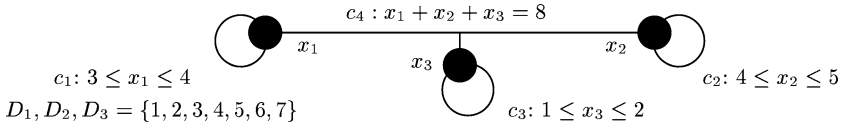

Fig. 3. (Boolean) course scheduling problem (2).

static CSP solution procedure to the whole updated problem. This is wasteful of the work done on the original problem. A solution to the updated Course Scheduling problem is $\{x_1 = 3, x_2 = 4, x_3 = 1\}$. The assignment to x_3 is common to both problems. An *rr*DCSP algorithm exploits this fact by focusing on the subset of variables whose assignments have become inconsistent in light of the changed problem structure. Hence, there is a significant efficiency saving made by avoiding re-solving a large part of the problem. This type of technique also offers the benefits of *stability*: there is as little disruption from one solution to the next as possible. An *rr*DCSP technique is more likely to achieve a reasonable level of stability since a static CSP technique essentially discards all the work done on the previous solution.

3. Fuzzy *rr*DFCSP

The combination of fuzzy CSP and *rr*DCSP to form fuzzy *rr*DFCSP enables the modelling of a changing dynamic environment, retaining the greater expressive power afforded by fuzzy CSP. In a dynamic environment where time may be limited, the ability of fuzzy CSP to produce the best current solution *anytime* will prove even more valuable. A fuzzy *rr*DFCSP can be thought of as a sequence of static flexible problems as per *rr*DCSP, with all possible changes being realised through restriction and relaxation. When applied to fuzzy CSPs, these operations can update a problem instance in more subtle ways. For example, the priority of a fuzzy prioritised constraint may change, as might the preferences of the individual tuples of a fuzzy preference constraint.

3.1. Fuzzy rrDFCSP solution techniques

The first solution technique examined for fuzzy *rr*DFCSP is based on the branch and bound (hereafter referred to as BB) approach to solving static fuzzy CSP [9]. It solves each instance in the dynamic sequence from scratch. However, the solution to the previous problem in the sequence is maintained and used to make savings. The solution to the previous problem in the dynamic sequence is checked against the current problem before it is solved. If the previous solution also constitutes a solution with $l_\top$ satisfaction degree for the current problem, no search is required. If it constitutes a solution for the current problem with a satisfaction degree l_i, such that $l_\bot < l_i < l_\top$, the solution is stored and used to set the necessary bound prior to search. During search, the solution to the previous problem is used to guide domain element selection for assignment. Given several possibilities with equivalent satisfaction degrees, an assignment is preferred which matches that of the solution to the previous problem, if possible. This method is a simplification of the oracles technique described in Section 2.3.

The second solution technique examined is a recently developed extension of the Local Changes (LC) local repair algorithm [35]. LC searches for a solution by resolving conflicts that occur when examining the solution to the previous problem in the sequence, within the context of the current problem. It divides the variable set X into three subsets: X_1, X_2 and X_3 (such that $X_1 \cup X_2 \cup X_3 = X$ and $X_i \cap X_j = \emptyset$ where $i, j \in \{1, 2, 3\},\ i \neq j$). X_1 is the set of variables with fixed assignments and is used to ensure termination of the algorithm;

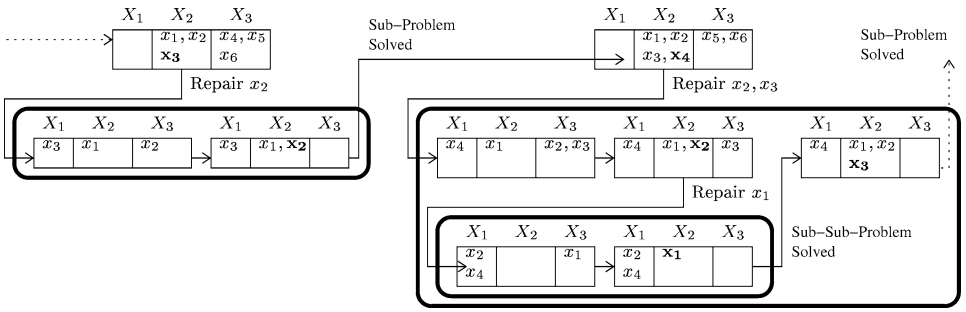

Fig. 4. The local changes algorithm: example search structure.

X_2 is the set of variables whose assignments are not fixed and X_3 is the set of variables currently without assignments.

Initially, only X_2 and X_3 are used. X_2 contains those variables that were also part of the previous problem in the sequence, and which therefore have assignments from the solution obtained to that problem. Hence, X_2 is the set of assignments that will be repaired to solve the current problem. X_3 contains variables that were not part of the previous problem, and which therefore do not yet have assignments. LC searches for consistent assignments to these variables. At the start of a problem sequence there is no previous problem, so X_2 is empty and X_3 contains all variables in the initial problem.

Consider the example in Fig. 4. The exact nature of the search depicted is unimportant, but note how the search process is controlled via the three variable sets. Starting from scratch or a partial assignment, LC selects and attempts to assign a value to $x_i \in X_3$ (the selected variable is shown in bold in the figure after its transfer from X_3 to X_2). If this assignment does not cause a conflict, LC selects another value from X_3 and so on. Otherwise, LC instigates a course of repairs with respect to the assignment to x_i, hence x_i is placed in X_1 to prevent a change to its assignment. The set of violated constraints is then examined and the subset of X_2 which is in conflict with x_i is unassigned. LC recursively solves the sub-problem of re-assigning this subset given the fixed assignment to x_i and the assignments to variables remaining in X_2. A sub-problem is denoted by a bounded area.

During the solution of one sub-problem further repairs may be necessary, in which case another variable is added to X_1 and so on. If no repair is possible (since one or more of the variables causing the conflict are in X_1) LC backtracks to solve the parent sub-problem in a different way. X_1 can be seen as a type of tabu list [19], preventing a potentially infinite cycle of repairs between mutually conflicting variables, ensuring termination.

Flexible Local Changes (FLC) [21], extends LC to support fuzzy constraints. The method of identifying conflicts and then focusing on the sub-problem of resolving those conflicts enables an effective use of the extra information afforded by fuzzy CSP in reducing search. FLC is structurally similar to LC in its use of the variable sets X_1, X_2 and X_3 to control the search. FLC searches for an optimal assignment, as opposed to an assignment that does not violate any hard constraint. FLC repeatedly selects a variable, $x_i \in X_3$ and assigns to it the domain element which results in the highest possible consistency given the set of currently assigned variables in $X_1 \cup X_2$. In making this

assignment repairs may be necessary to elements of X_2. Termination is upon finding an optimally consistent complete variable assignment.

The present work considers finding optimal solutions of fuzzy *rr*DFCSPs directly using min aggregation. An alternative approach is to solve each problem in a dynamic sequence via iteratively resolving successive classical CSPs, constructed by allowing all constraint tuples with consistency degree greater than a prescribed α level. An optimal solution can be found by moving α from $l_\perp$ upwards one degree at a time up to the level where the CSP becomes inconsistent. Where $|L|$ is large, a binary search could be adopted to set α [6,10]. This remains an important avenue of future work, as is the question of how best to incorporate dynamicity into this method.

3.2. Fuzzy arc consistency

The idempotent *min/max* operators employed by fuzzy constraint satisfaction enable the straightforward support of consistency enforcing techniques [30]. Fuzzy arc consistency [9] can significantly enhance the performance of both BB and FLC. Fuzzy arc consistency holds if any assignment involving one variable with a satisfaction degree $l_i \in L$ can be extended to any 2 variables, maintaining a satisfaction degree of l_i.

Enforcing fuzzy arc consistency is essentially the same process as for classical CSP [17]. The FAC3( ) procedure [9], based on classical AC3( ) [17], is used. A queue of arcs from the constraint network is maintained. An *arc*(x_i, x_j) is selected and *revised*, i.e., the consistency degree of each $d_i \in D_i$ is updated according to:

$$cons(x_i = d_i) \otimes \max_j \big(cons(x_j = d_j) \otimes cons(x_i = d_i, x_j = d_j)\big).$$

If the consistency degree of any $d_i \in D_i$ is updated by this revision, further revisions may be possible to variables related to x_i by constraints. Hence each *arc*(x_h, x_i), where $h \neq j$, is added to the queue (if not already present). The algorithm terminates when the queue is empty. The complexity of FAC3( ) is $O(\ell m^3 e)$ [9], where $\ell = |L|$ and e is the number of arcs in the network. The upper bound, β, on the consistency of the whole problem is the minimum over all the variables of the maximum consistency degrees of their domain elements.

A fuzzy version of the classical AC4( ) arc consistency algorithm has also been developed, with complexity $O(m^2 e)$ [6]. However, although AC3( ) has a poorer worst case time complexity than AC4( ), it is often preferred in practice because of its better average case performance [36], hence the choice made here. It is of course important to test the performance of FAC4( ) in practice, and this is an important item of future work.

Pre-processing a problem with fuzzy arc consistency provides a bound, β, on its consistency. Both BB and FLC benefit from β since it facilitates early termination when a solution of satisfaction degree of β is found. Furthermore, when FLC examines the previous solution with respect to the current problem it is only necessary to repair constraints whose consistency is below β, leading to a higher percentage of solution reuse. Without β all constraints whose consistency degree is less than $l_\top$ are repaired, regardless of the fact that a solution of consistency $l_\top$ may not exist. FLC may also employ β when considering which constraints to repair during search.

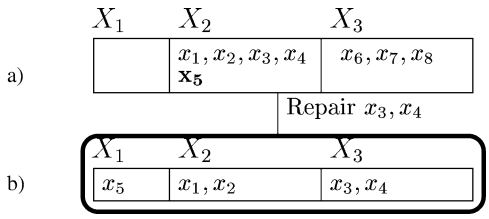

Fig. 5. Sub-problem composition under repair.

It is also possible to enforce various levels of arc consistency during search. In the case of branch and bound, whose search proceeds chronologically, the assignment and unassignment of variables proceeds in an ordered manner such that it can be guaranteed that a particular set of assignments is fixed for the duration of a sub-problem. Consistency is enforced with respect to the current variable assignments at each search node, enabling the calculation of a more accurate upper bound, β, on the satisfaction degree of the remaining sub-problem. As per basic BB, if this bound does not exceed α, the satisfaction degree of the best solution found thus far, this search branch can be pruned.

FLC's search does not proceed chronologically: all variables in X_2 can be unassigned in any order to effect repairs. Hence, it is difficult to enforce consistency based on assignments to elements of X_2 as changes made by such propagation must be undone upon unassignment. This is a potential weakness which will be tested in the following section. The solution proposed for LC (and adopted for FLC) is to enforce consistency with respect to X_1 [35]: it *is* guaranteed that assignments to X_1 are fixed for the duration of a sub-problem, hence they cannot be undermined by repair. Consider Fig. 5(a). If $x_5 \in X_3$ is selected and instantiated and the current assignments to x_3 and x_4 are found to be inconsistent, the sub-problem of resolving these inconsistencies is depicted in Fig. 5(b). Since only variables $\{x_1, x_2, x_3, x_4\}$ can possibly be present in the sub-problem, it is at first sight only worth filtering the domains of these variables with respect to an assignment to x_5. Generally, when considering an assignment to x_i, the domains of the current elements of X_2 are updated through consistency enforcing because it is the elements of X_2 which are unassigned to be repaired in the sub-problem for which x_i is fixed.

To combat the problem of FLC's reduced constraint propagation ability, bounds information can be extracted from the elements of X_3 (x_6, x_7, x_8 in the example). Although these variables will not feature in the repair sub-problem, they *do* have to be assigned eventually, and their assignments may interact with the variables under consideration for repair. Consider again the example shown in Fig. 5. If consistency enforcing showed, for example, that it was always the case that the current assignment to x_5 precluded any assignment to x_6 because of some constraint $c(x_5, x_6)$, then there is little point in resolving the conflicts with x_1 and x_2. In general, constraint propagation using the elements of X_3 can contribute in this way to the calculation of β for the repair sub-problem.

3.3. The deletion threshold

Once a necessary bound for a sub-problem is established, it can be exploited by consistency enforcing hybrids of both BB and FLC. Given, for example, a necessary bound

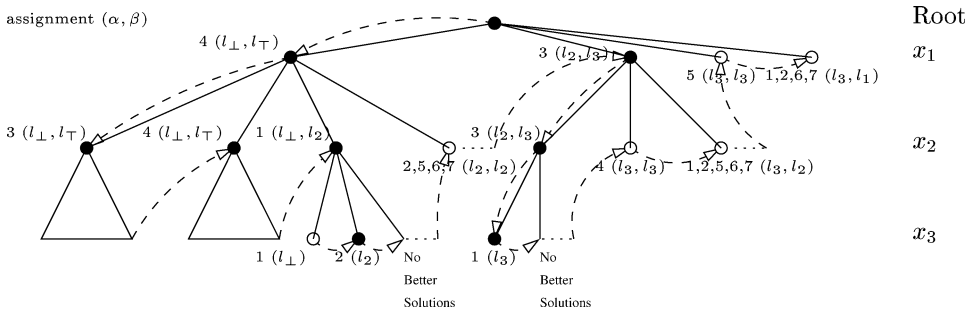

Fig. 6. Branch and bound solution to the problem shown in Fig. 2.

of $l_i \in L$, all domain elements that can be shown to have a unary satisfaction degree of $l_j \in L$ such that $l_j \leqslant l_i$ can be deleted for the duration of the sub-problem. The justification for this comes directly from the conjunction of satisfaction degrees implemented using the *min* operator: any single assignment with a satisfaction degree of l_i or less can never form part of a solution to the sub-problem with a satisfaction degree of greater than l_i.

The ability to delete domain elements early saves further revision of the deleted elements. Otherwise, each of these elements could potentially be revised i more times before reaching $l_\perp$ and being deleted. Furthermore, constraint propagation would not necessarily reduce the satisfaction degrees of the affected elements to $l_\perp$. Hence, they might never be deleted, but their presence degrades heuristics based on selecting variables with the smallest domain first, whose use is popular in constraint satisfaction [22].

3.4. Solution of the fuzzy course scheduling problem

The BB approach solves the Fuzzy Course Scheduling problem as shown in Fig. 6, where a triangle denotes a sub-tree in which the algorithm failed to find a solution better than the best currently known. Since this is the first problem in the sequence, the problem is treated as a static fuzzy CSP. In the figure, the domain element heuristic prefers the assignment with the highest satisfaction degree. The algorithm rapidly finds the solution $\{x_1 = 4, x_2 = 1, x_3 = 2\}$ with the satisfaction degree l_2. This bound on further solutions allows it to prune effectively the search tree subsequently. Hence, the optimal solution $\{x_1 = 3, x_2 = 3, x_3 = 1\}$, with satisfaction degree l_3, is quickly found. FLC uses the same domain element selection heuristic as BB. The solution procedure is shown up to the point that an optimal solution is found:

- Select x_1. Relevant constraint: $c_1(\mathbf{x_1})$.
  - $x_1 \leftarrow 4$, consistency $= l_\top$.
  - $X_1 = \{\}, X_2 = \{x_1\}, X_3 = \{x_2, x_3\}$.
- Select x_2. Relevant constraint: $c_2(\mathbf{x_2})$.
  - $x_2 \leftarrow 3$, consistency $= l_\top$.
  - $X_1 = \{\}, X_2 = \{x_1, x_2\}, X_3 = \{x_3\}$.
- Select x_3. Relevant constraints: $c_3(\mathbf{x_3}), c_4(x_1, x_2, \mathbf{x_3})$.
  - For all assignments, consistency $= l_\perp$.

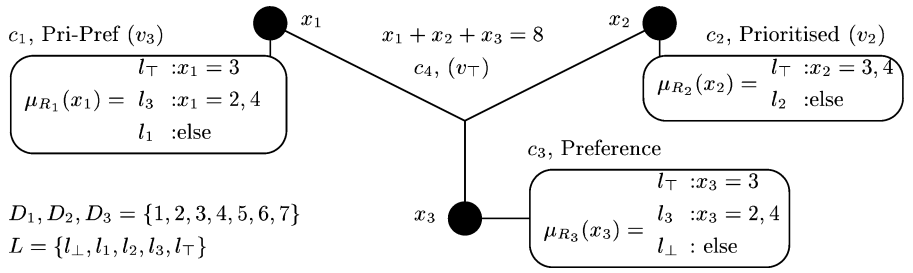

Fig. 7. Fuzzy course scheduling problem—part 2.

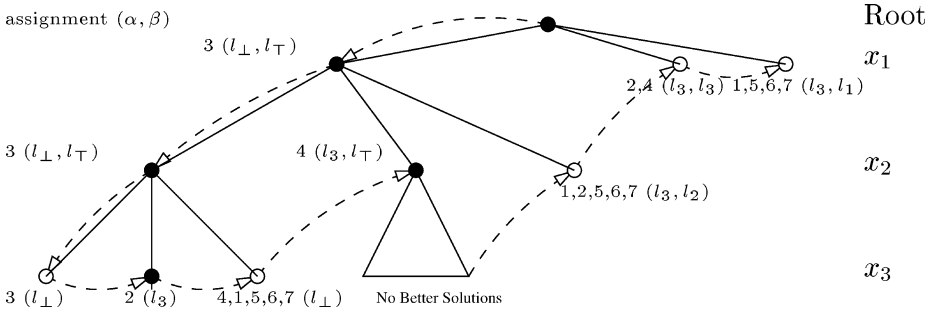

Fig. 8. Branch and bound solution to the problem shown in Fig. 7.

- Start repair from $x_3 \leftarrow 1$ (best local assignment).
- Repair $c_4(x_1, x_2, x_3)$: Choose x_1.
- $X_1 = \{x_3\}$, $X_2 = \{x_2\}$, $X_3 = \{x_1\}$.

• Select x_1. Relevant constraints: $c_1(\mathbf{x_1})$, $c_4(\mathbf{x_1}, x_2, x_3)$.
- $x_1 \leftarrow 3$, consistency $= l_3$.
- Assignment $\{x_1 = 3, x_2 = 3, x_3 = 1\}$, consistency $= l_3$, is optimal.

The next time the course is run (see Fig. 7), it is expanded to 8 sessions (c_4) and the preference is now for a greater number of training sessions (c_3). Additionally, other commitments mean that Professor A prefers to give fewer lectures (c_1). The solution procedures for both algorithms on this updated problem are shown below, illustrating the savings that can be achieved through the maintenance of effort from previous problems.

BB first examines the solution to the previous problem with respect to this new problem. Since it violates the hard constraint and therefore has a satisfaction degree of $l_\perp$, it does not help in setting a necessary bound on the satisfaction degree of any new solution. However, the previous solution *is* used during domain element selection: the assignments $x_1 = 3$ and $x_2 = 3$ are made because they match the assignments to those variables in the previous solution. The optimal solution $\{x_1 = 3, x_2 = 3, x_3 = 2\}$ is then found with little effort (Fig. 8). FLC's examination of the solution to the previous problem reveals that constraints c_3 and c_4 are violated. Since these constraints share a variable, x_3, it is possible that by reassigning just this one variable, both constraints can be satisfied. This proves to be the case:

- Evaluate assignment $\{x_1 = 3, x_2 = 3, x_3 = 1\}$: consistency $= l_\perp$.
  - Repair $c_3(x_3)$, $c_4(x_1, x_2, x_3)$: Choose x_3 to cover both.
  - $X_1 = \{\}$, $X_2 = \{x_1, x_2\}$, $X_3 = \{x_3\}$.
- Select x_3. Relevant constraints: $c_3(\mathbf{x_3})$, $c_4(x_1, x_2, \mathbf{x_3})$.
  - $x_3 \leftarrow 2$, consistency $= l_3$.
  - Assignment $\{x_1 = 3, x_2 = 3, x_3 = 2\}$, consistency $= l_3$, is optimal.

For brevity of presentation, the steps performed by FLC to verify that this is indeed an optimal solution are not given.

4. An empirical study of fuzzy *rr*DFCSPs

This section describes an empirical study of random fuzzy *rr*DFCSPs. Due to the size of the study (almost 40,000 individual instances), selected results are presented to reflect the general trend of the data. The problems are sequences of ten fuzzy CSP instances. Binary constraints are considered, following the approach of many studies of classical CSP [13, 28]. Each instance is generated using the following parameters:

- n. Number of variables, taking values from $\{20, 30, 40\}$ in order to gauge the effects of increasing problem size.
- m. Domain size, fixed at 6 throughout to limit the size of this study. Further studies should examine the effects of varying m.
- $\ell = |L|$. Takes values from $\{3, 4, 5\}$ to examine the effects of an increasing amount of flexibility on the difficulty of finding the best solution.
- *con*. Connectivity of the constraint graph, the proportion of the variables which are related by a constraint. Takes values from $\{0.25, 0.5, 0.75\}$, allowing an examination of the effects of an increasingly connected graph.
- t. Constraint tightness, the proportion of assignment combinations that are disallowed by each constraint. Takes values from $\{0.1, 0.2, \ldots, 0.8, 0.9\}$.

The granularities chosen for the connectivity and tightness parameters make this study feasible. Interesting phenomena may occur between the values chosen, which future studies might investigate. Constraints are divided into an equal number of fuzzy prioritised, preference, and prioritised-preference types. Priority degrees are evenly distributed amongst the prioritised and prioritised-preference constraints. Similarly, preference degrees are evenly distributed amongst the tuples of each preference and prioritised-preference constraint.

Random restriction/relaxation operations are performed. A further parameter, *ch*, determines the proportion of constraints that are removed and replaced (when *ch* = '1 constraint', a single constraint is replaced between instances). To maintain a uniform constraint graph connectivity, the same number of constraints are added as are taken away. Each constraint removed is replaced by a constraint of the same type and tightness. If it is a prioritised or prioritised-preference constraint, the priority assigned is the same as the removed constraint to maintain the original distribution. The justification for the effort of

maintaining a uniform problem structure is to eliminate the effects of differing proportions of fuzzy constraint types or priority/preference distributions.

Five dynamic sequences of ten instances were generated per parameter combination. Mean results are reported, hence each point on the graphs presented represents the solution of 50 instances. In all, 3645 dynamic sequences were generated, totalling 36450 individual problem instances to be solved.

4.1. The algorithms studied and evaluation criteria

The performances of basic BB and FLC were so poor compared to refined versions that they were not chosen for inclusion in this study. The naming convention for the variants tested is as follows. An 'AC' prefix denotes that fuzzy arc consistency is established once prior to search. If the suffix is 'FC', then the consistency enforcing process corresponds to classical Forward Checking [15]: the domains of unassigned variables are revised once with respect to each new assignment. Similarly, if the suffix is 'FMAC' then, as per the classical MAC algorithm, fuzzy arc consistency is established initially and with respect to each new variable assignment. Unless otherwise stated, the deletion-threshold is used to enforce fuzzy arc consistency. A comparison with dynamic backtracking [14] is omitted since, as recognised by [35], this algorithm has strong similarities with LC. Updating dynamic backtracking to support dynamic CSPs and flexible constraints would produce a very similar algorithm to FLC.

The variants of BB tested are: BBFC, ACBBFC, BBFMAC. A similar number of basic variants of FLC are also tested. In addition, three extra hybrids are tested which also take into account the domains of variables not immediately in line for repair, as detailed in Section 3.2. These are denoted by the suffix 'ft' (for 'full test'): FLCFC, FLCFCft, ACFLCFC, ACFLCFCft, FLCFMAC, FLCFMACft. Performance is judged by three criteria:

- *Constraint checks*. Every time a constraint is queried for the consistency degree of a pair of assignments, this is counted as a constraint check.
- *Search nodes* expanded. Every assignment of $d_i \in D_i$ to x_i corresponds to a node in the search tree.
- *Solution stability*. Since, in this study, the set of variables does not change within a dynamic sequence, the stability of the solution to each problem as compared with the last can easily be measured as the proportion of assignments that are the same. It is appropriate to use this measure, since there may be a number of solutions with optimal satisfaction degrees.

Run-times are not reported, although they have been observed to track consistency checks. Due to the size of the study, results were collected over a large network of machines of varying capabilities and under differing loads. It would be difficult, therefore, to place any great faith in run-time results.

4.2. Heuristics investigated

Three operations which are ordered heuristically are the selection of a variable to instantiate next, the choice of domain element to instantiate, and the order in which constraints are checked. Apart from those presented here, a variety of other heuristics were tested extensively (see [23] for details).

As the analogue of classical Backtrack, it is simple to use the Brelaz heuristic [4] with BB. That is, to select the variable with smallest remaining domain first, breaking ties by preferring the variable with the greatest connectivity to the currently unassigned variables. FLC may also use a Brelaz-type heuristic, but the search structure of this algorithm, controlled by the sets X_1, X_2 and X_3 provides the possibility of creating novel variations on the variable selection heuristic. The heuristic employed in this study, `SmallestD-`X_{23}, starts by selecting the variable with the smallest remaining domain and breaks ties using the connectivity of the variables in $X_2 \cup X_3$, i.e., all those variables whose assignments are free to change. Note that even though elements of X_3 do not figure in the sub-problem of repairing assignments, they do have to be assigned eventually and hence interact with the current variable.

The domain element selection heuristic is the main method by which the BB algorithms make use of dynamic information. In order to attempt to find the best solutions first, the element with the highest consistency degree is chosen first. However, if there are several such elements, ties are broken by examining the solution to the previous problem in the sequence, and if one of the tied elements matches the assignment in the previous solution, it is chosen.

For FLC, value selection is split into two cases, where repairs are and are not necessary. Taking the latter case first, the element with the highest consistency degree is selected first to find the solution with the highest consistency degree earlier. As per BB, ties are broken by assigning the same element to a variable as appeared in the previous solution. This may appear counter-intuitive, but a conflict commonly necessitates the unassignment of multiple variables. Re-assigning each of these may trigger further repairs, and as a side effect remove conflicts with some previous solution assignments. If so, it is possible to reinstate the previous solution assignments, promoting stability.

When repairs are necessary, it is intuitive to select a repair that is most likely to succeed. The domain element selection heuristic used for repair, `ConsDeg-PrevSoln-`*con*, follows the approach of the BB algorithm to an extent in first breaking ties by matching against the previous solution. Ties are broken further by preferring to repair the set of variables with the least connectivity to the remainder of the problem.

The process of checking constraints is ordered such that the constraint most likely to fail is checked first. This is achieved by examining the size of the domains D_i, D_j at the end of each $arc(x_i, x_j)$ to be checked. The heuristic prefers those arcs with the smallest summed domain size. Ties are broken by restricting the comparison to each D_i, since this is the domain being filtered.

4.3. Results: search effort

This section investigates the relative search effort required by each algorithm to solve the set of random fuzzy *rr*DFCSPs. The three variants of BB are tested against three

hybrids of FLC. The 'ft' (see Section 4.1) variants hold a small but consistent advantage over those variants of FLC that do not perform constraint propagation on elements of X_3. For clarity, therefore, only representative results for the 'ft' variants are presented in Figs. 9–16.

Throughout the results the forward-checking algorithms require many fewer constraint checks than the algorithms maintaining fuzzy arc consistency. The increased cost of enforcing fuzzy arc consistency (see Section 3.2) is a contributing factor to this result. In terms of search nodes the roles are reversed, with the arc consistency-maintaining algorithms performing significantly better. This is unsurprising: by performing more constraint propagation, these algorithms can detect dead ends in the search tree more quickly. BBFC dominates FLCFCft by both search cost measures. This is as predicted in Section 3.2, with the more rigid chronological search structure of BB allowing greater propagation.

ACBBFC and ACFLCFCft typically require fewer constraint checks than BBFMAC and FLCFMACft respectively. The MAC algorithms fail to exploit the extra propagation gained from maintaining fuzzy arc consistency throughout search to the extent that the overall number of constraint checks is reduced. The reverse is true of search nodes, as would be expected: the forward checking algorithms are unable to detect dead ends as early as the MAC variants of both BB and FLC.

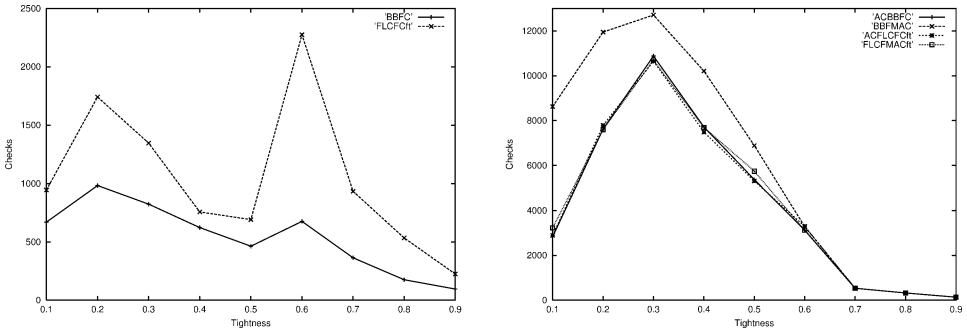

Fig. 9. Mean constraint checks: $\ell = 3$, $ch =$'1 constraint', $n = 20$, $con = 0.25$.

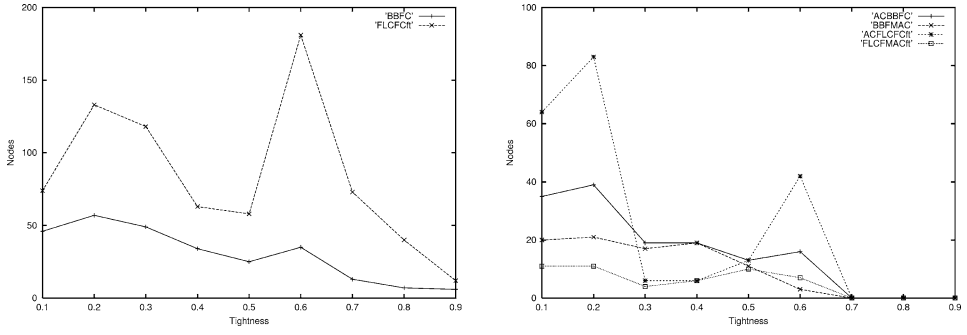

Fig. 10. Mean nodes: $\ell = 3$, $ch =$'1 constraint', $n = 20$, $con = 0.25$.

A comparison of ACBBFC versus ACFLCFCft and BBFMAC versus FLCFMACft reveals that, on easier, more flexible problems the FLC variants both require fewer constraints checks and explore a smaller search tree. This situation changes as more difficult problems are considered (see Figs. 13 and 14), when the greater propagation enabled by chronological search results in a large reduction in overall effort.

A prominent feature in the results is the presence of multiple phase transitions. This is predictable, with the number of phase transitions corresponding to the number of degrees of consistency available above $l_\perp$. Transitions exist as t increases from problems that are solvable to a consistency degree, l_a, to being unsolvable to l_a yet still solvable to the adjacent consistency degree below l_a in L. The phase transitions, corresponding to peaks in search effort, are particularly evident when $\ell = 5$ (see Figs. 11 and 12). The final phase transition, from a problem solvable with consistency degree l_1 to an unsolvable problem, corresponds to the single phase transition widely observed in Boolean problems [28].

Within the constraint check results on easier problems, phase transitions are most clear for those algorithms that perform an initial fuzzy arc consistency step, but not for those that do not. The cost, in constraint checks, of enforcing fuzzy arc consistency initially (as noted, a more expensive process than Boolean arc consistency) dominates the cost of the rest of the search. The peak observed for the pre-processing algorithms is therefore indicative of a phase transition in the cost of enforcing fuzzy arc consistency also observed

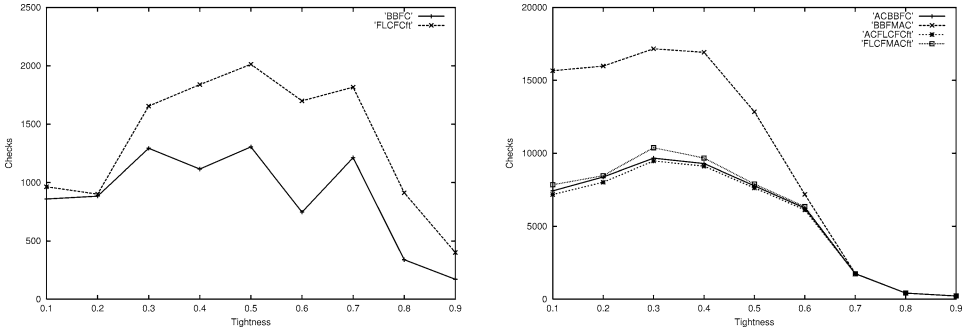

Fig. 11. Mean constraint checks: $\ell = 5$, $ch =$'1 constraint', $n = 20$, $con = 0.25$.

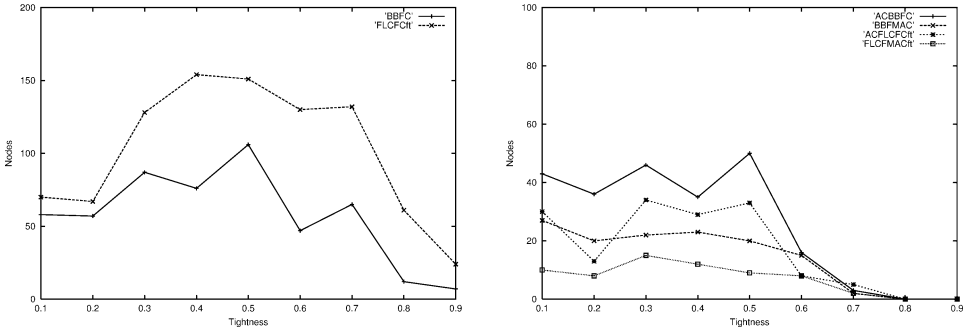

Fig. 12. Mean nodes: $\ell = 5$, $ch =$'1 constraint', $n = 20$, $con = 0.25$.

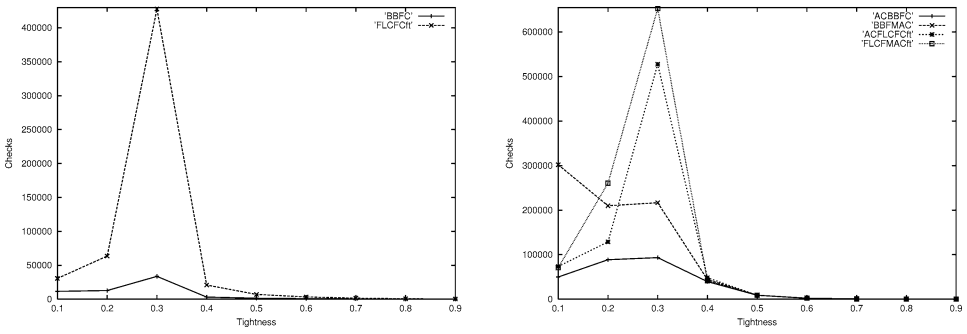

Fig. 13. Mean constraint checks: $\ell = 4$, $ch = 0.1$, $n = 40$, $con = 0.5$.

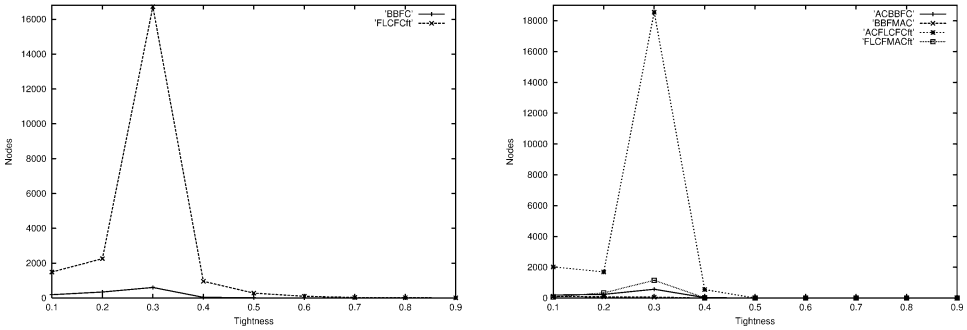

Fig. 14. Mean nodes: $\ell = 4$, $ch = 0.1$, $n = 40$, $con = 0.5$.

in Boolean problems [12]. A single peak might be explained by the fact that enforcing fuzzy arc consistency, as opposed to searching for a solution, is centred around constraint propagation. When the constraints are less tight, independent of the number of consistency degrees, less propagation is possible, hence enforcing fuzzy arc consistency costs less. Conversely, when the constraints are tighter, strong propagation is possible, revising the unary consistency degree of many domain elements rapidly. Between these two cases, revisions are possible, but in small amounts, resulting in the need to check many more constraints before fuzzy arc consistency is enforced. Multiple phase transitions are not obscured when search nodes results are considered. These reflect only the search process, the cost of the pre-processing step having been factored out.

Increasing n, con and (to a lesser extent) ℓ produces problems that are in general significantly more difficult to solve. Enforcing fuzzy arc consistency is no longer the dominant cost in terms of constraint checks. Thus, a more uniform behaviour pattern is observed for each algorithm. The fact that a single transition peak is visible in results such as those in Figs. 13 and 14 is due to the resolution of the tightness axis: the high connectivity means that the phase transitions occur more quickly as the tightness parameter is increased.

Search effort increases with the proportion of change between problem instances for all algorithms. This trend is indicative of the extra effort required to maintain a stable solution

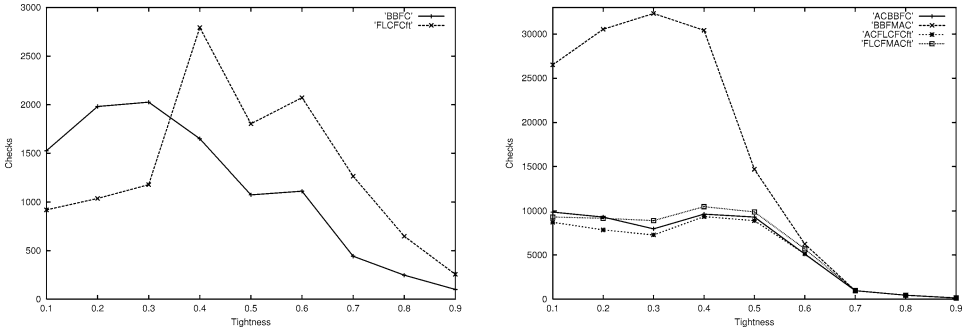

Fig. 15. Mean constraint checks: $\ell = 4$, $ch = 0.25$, $n = 20$, $con = 0.25$.

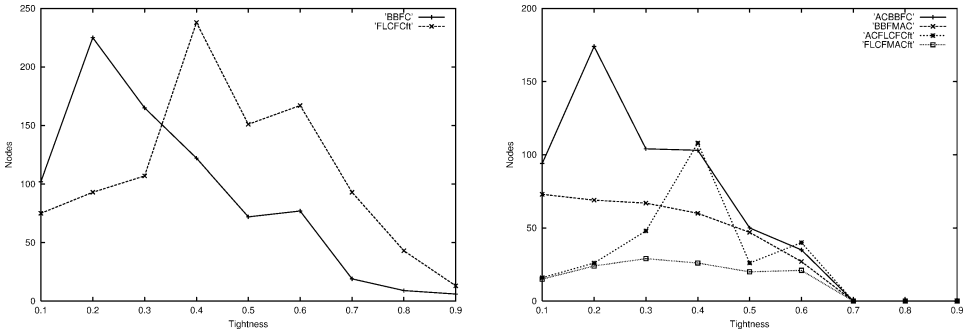

Fig. 16. Mean nodes: $\ell = 4$, $ch = 0.25$, $n = 20$, $con = 0.25$.

in the face of an increasingly dynamic problem structure. The comparative performance of the BB and FLC algorithms is skewed as *ch* increases. The repair-based strategy of the FLC variants appears to be most effective when constraints are loose and a targetted repair of a relatively small sub-problem is enough to produce an optimal solution. BB does not attempt repairs and so is unable to focus on small areas of the problem. Forcing the 'wrong' stable assignment early in the search can cause a significant increase in overall effort. As t increases, however, the stronger propagation of BB begins to tell, revealing poor assignment choices earlier.

4.4. Results: stability

This section examines the ability of variants of BB and FLC to maintain stable solutions, i.e., to minimise the difference between consecutive solutions to problems in the dynamic sequence. Although stability is not an explicit optimisation criterion, it is expected that FLC's method of repairing the previous solution and the domain element selection heuristics of both BB and FLC will produce stable solutions. In future, preservation of stability could be added as a means of breaking ties between solutions with optimal consistency degrees, although at a greater computational cost. Figs. 17–20 present representative results. Entries corresponding to the highest tightness values are typically

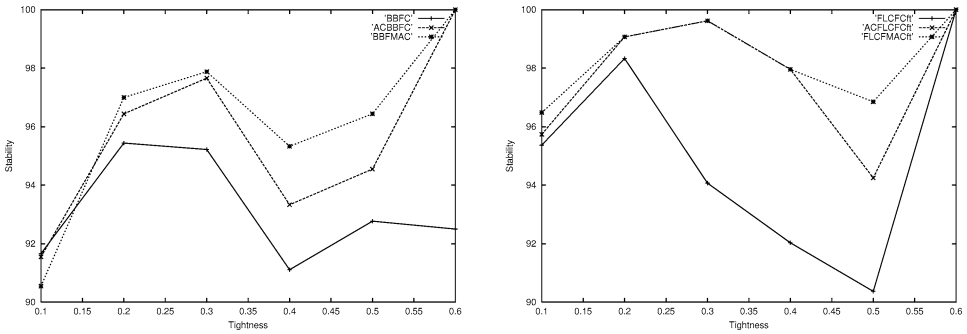

Fig. 17. Stability: $\ell = 3$, $ch =$'1 constraint', $n = 20$, $con = 0.25$.

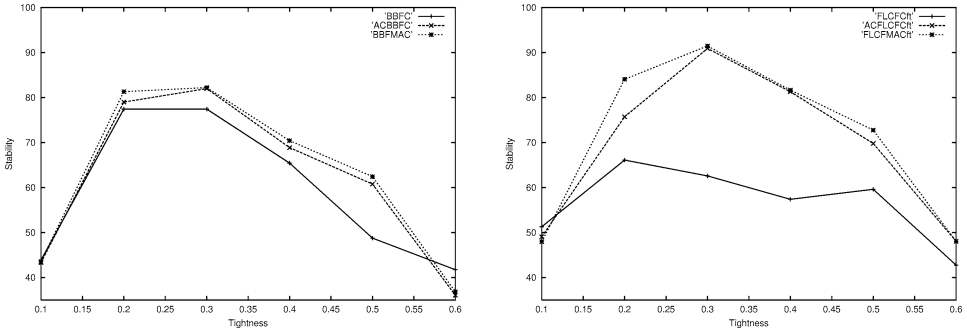

Fig. 18. Stability: $\ell = 3$, $ch = 0.25$, $n = 20$, $con = 0.25$.

based on stability results for fewer problems since this entry coincides with the final phase transition to unsolvable problems. In this case, stability is measured between the previous solvable problem and the current solvable problem, rather than simply between adjacent problems in the sequence.

The variants of BB and FLC that enforce fuzzy arc consistency, whether as a pre-processing step or during search consistently produce more stable solutions than forward checking versions. Enforcing fuzzy arc consistency provides a bound, β, on the remaining problem (Section 3.2). In the case of FLC, constraints already satisfied to consistency degree at least β need not be repaired, aiding stability by not unnecessarily disturbing assignments. Similarly for BB, a stable assignment of at least consistency degree β can be preferred over another with a higher consistency degree: in all circumstances the overall problem cannot have a consistency degree exceeding β.

Given the utility of a more informed bound on the remaining problem, it is unsurprising that BBFMAC and FLCFMACft produce, almost uniformly, the most stable solutions of the BB and FLC variants respectively. Enforcing fuzzy arc consistency appears to be particularly important to the stability of FLC. Without this, FLCFCft produces less stable solutions than BBFC, whereas ACFLCFCft and FLCFMACft maintain a consistent advantage over their BB counterparts. The advantage of the FLC algorithms is also a result of the type of search they employ. Not only are stable assignments preferred at all choice

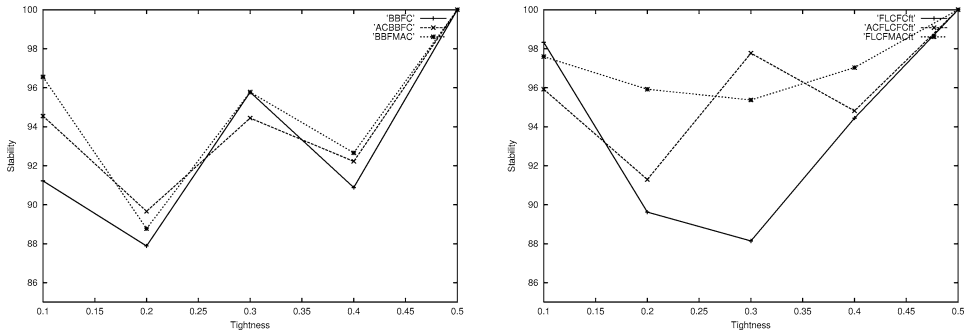

Fig. 19. Stability: $\ell = 5$, *ch* = '1 constraint', $n = 20$, $con = 0.5$.

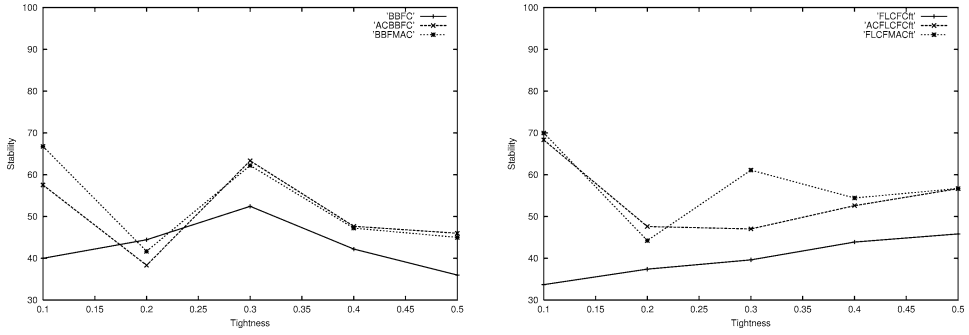

Fig. 20. Stability: $\ell = 5$, $ch = 0.25$, $n = 20$, $con = 0.5$.

points, but FLC's repair-based search actively seeks to leave in place assignments that match the previous solution.

As expected, the ability of all algorithms to produce stable solutions decreases as the amount of change between instances increases (Figs. 18 and 20). Varying ℓ or n affects overall stability levels since a less flexible problem structure allows less freedom to find a stable solution. However, varying these parameters does not change the relative performance of the algorithms tested.

4.5. Utility of dynamic information

BB and FLC make extensive use of information stored in the solution to the previous problem in a dynamic sequence to improve the efficiency of searching for a solution to the current problem. This section compares 'crippled' variants of BB and FLC that do not have access to dynamic information against their fully dynamic counterparts. In the case of BB, this means that it no longer has a heuristic 'guide' when making variable assignments, and FLC must solve each new problem instance from scratch. The algorithms chosen for this part of the study are BBFC, BBFMAC, FLCFCft and FLCFMACft. The suffix 'noDyn' indicates those algorithms whose dynamic capability has been removed. Representative results are presented in Figs. 21–24.

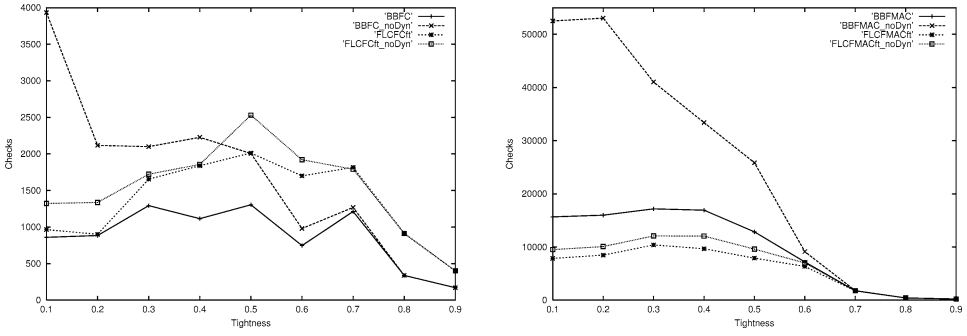

Fig. 21. Mean constraint checks: $\ell = 5$, *ch* = '1 constraint', $n = 20$, $con = 0.25$.

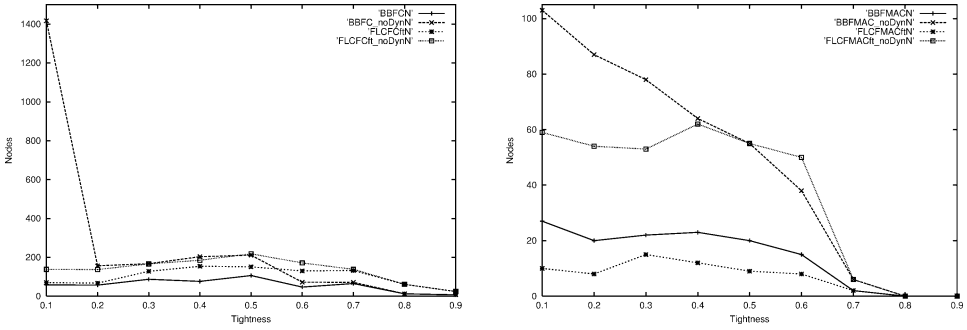

Fig. 22. Mean nodes: $\ell = 5$, *ch* = '1 constraint', $n = 20$, $con = 0.25$.

With respect to constraint checks (Fig. 21), BBFMAC suffers the most from losing the dynamic capability. This algorithms appears to use dynamic information to target the greater amount of propagation it does most effectively. For the other algorithms the benefit of making use of dynamic information is less pronounced in terms of constraint checks. The number of search nodes by each algorithm, however, *is* significantly affected (Fig. 22). The use of dynamic information allows each algorithm to make a more informed exploration of the search tree, hence reducing its overall size.

Unsurprisingly, the advantage of trying to re-use information from the solution to the previous problem is eroded as the proportion of the problem which changes from instance to instance is raised. A clear advantage is still evident even when fully one quarter of the problem changes between instances. Also, as constraint tightness increases to the point that the problems are unsolvable, dynamic information is no longer helpful to reduce search effort. Nor does it hinder search, however, which might have been the case if the domain element selection heuristics had made poor choices with respect to the effort to prove unsolvability. The number of consistency degrees, ℓ, has minimal effect on the size of the advantage gained by using dynamic algorithms.

Stability results are presented in Figs. 23 and 24. When the proportion of change between instances is very small, the non-dynamic algorithms find relatively stable solutions simply because the problem structure has not changed sufficiently to affect their

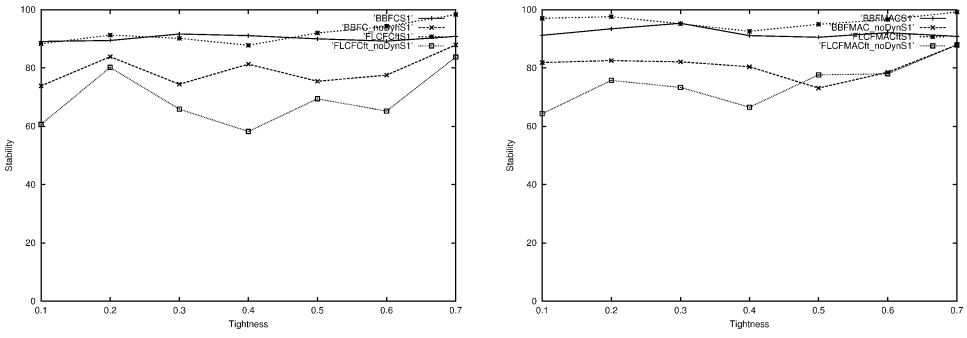

Fig. 23. Stability: $\ell = 5$, $ch =$'1 constraint', $n = 20$, $con = 0.25$.

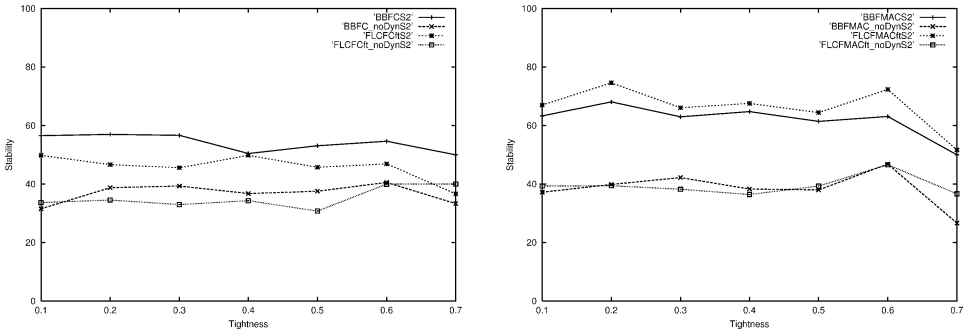

Fig. 24. Stability: $\ell = 5$, $ch = 0.25$, $n = 20$, $con = 0.25$.

heuristics. Even so, the dynamic algorithms use the extra information available to produce significantly more stable solutions. As expected, the ability to find stable solutions is diminished as the proportion of the problem that changes between instances is increased. However, the dynamic algorithms are still able to maintain a large advantage over their uninformed counterparts.

4.6. Utility of the deletion threshold

Section 3.3 described the deletion threshold, which allows consistency-enforcing hybrids to filter domain elements as early as possible. This method was tested on BBFC, BBFMAC, FLCFCft and FLCFMACft with representative results presented in Figs. 25 and 26. The suffix 'noDel' indicates algorithms with deletion threshold $l_\perp$. Overall, the utility of the deletion threshold increases with ℓ since an ever greater number of revisions are otherwise required to revise the consistency degree of a domain element to $l_\perp$. At high connectivities more constraint propagation is possible, increasing the likelihood that the consistency degree of a domain element is revised to $l_\perp$ and decreasing the utility of the deletion threshold.

Most benefit is gained in terms of constraint checks by algorithms maintaining fuzzy arc consistency as they perform more constraint propagation. In terms of search nodes,

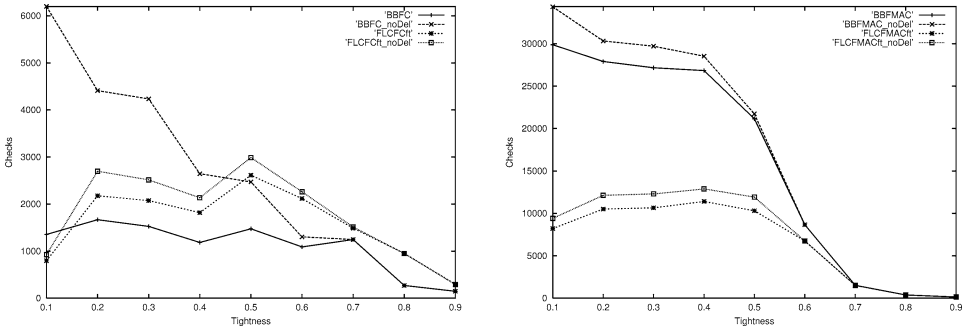

Fig. 25. Mean constraint checks: $\ell = 5$, $ch = 0.1$, $n = 20$, $con = 0.25$.

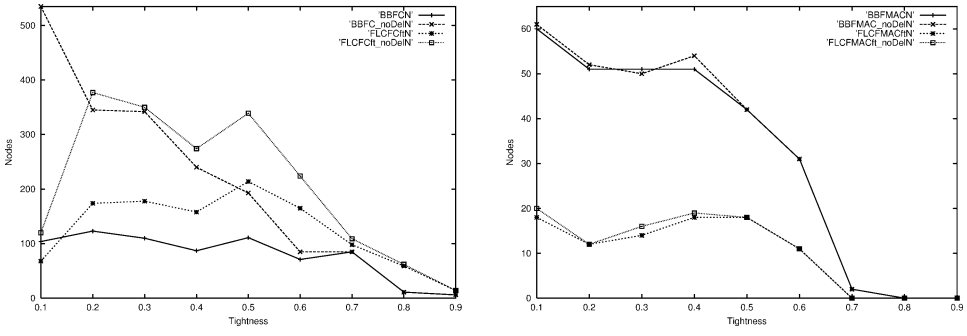

Fig. 26. Mean nodes: $\ell = 5$, $ch = 0.1$, $n = 20$, $con = 0.25$.

however, the forward checking algorithms are most improved. Forward checking is weak propagation, hence the ability to filter more domain elements has greater impact. In turn, filtering domain elements early can improve the performance of variable ordering heuristics (see Section 3.3).

4.7. Summary of empirical results

One of the most important conclusions that can be drawn from these results is that the use of dynamic solution techniques is justified: effort required to maintain and re-use information from the previous problem in a dynamic sequence is compensated for by an increase in the quality of results as measured by all three criteria given in Section 4.1. However, it is difficult to choose a clear 'winner'. Although the forward checking algorithms consistently gave better results in terms of constraint checks, they were often poorer when search nodes were considered. Conversely, the algorithms that maintain fuzzy arc consistency consistently explored fewer search nodes, but made many more constraint checks. One general rule is that the BB algorithms are more efficient, as predicted, on the harder problems, since their more rigid search structure allows more effective constraint propagation. Another is that the FLC variants produce more stable solutions since they

actively work to maintain the assignments in the solution to the previous problem in an instance rather than simply re-assigning the same domain element as a tie-breaker.

5. Flexible planning problems

Many real-world planning problems present the need for soft constraints. Consider an example from the logistics domain where a valuable package must be loaded onto a truck. The preconditions of the `Load-truck` action state that (i) the truck and (implicitly) valuable package must be co-located, and (ii) a guard must be present. While precondition (i) is imperative, precondition (ii) is a preference or soft constraint and can be relaxed with an associated damage to the resultant plan. Classical AI Planning [1] is too rigid to model such problems, being founded on hard constraints, and hence has no capability of making compromises. A *flexible planning problem* is introduced to allow a tradeoff between plan length and the compromise decisions made. Fuzzy CSP is the formal foundation underlying the definition of a flexible planning problem. Subjective truth degrees are associated with propositions while satisfaction degrees are associated with plan operators and goals, expressing how well their preconditions are satisfied by a set of flexible propositions. Plan satisfaction degrees are calculated from the satisfaction degrees of their constituent instantiated operators and goals, enabling a direct comparison amongst a number of plans containing different compromises.

More formally, a flexible planning problem, Ψ, consists of a 4-tuple, $\langle \Phi, \mathrm{O}, \mathrm{I}, \Gamma \rangle$, denoting sets of plan objects, flexible operators, initial conditions consisting of flexible propositions, and flexible goal conditions. Boolean propositions are herein replaced by flexible propositions, ρ, of the form $(\rho\ \phi_1, \phi_2, \ldots, \phi_j\ k_i)$, where each $\phi_j \in \Phi$ and k_i is an element of a totally ordered set, K, which denotes the subjective degree of truth of the proposition. K is composed of a finite number of membership degrees, $k_\perp, k_1, \ldots, k_\top$. The original Boolean proposition type is captured at the end points of K, with $k_\perp \in K$ and $k_\top \in K$ indicating total falsehood and total truth respectively. For brevity, when dealing with propositions which only ever take a truth value of $k_\perp$ or $k_\top$, the Boolean style of $\neg(\rho\ \phi_1, \phi_2, \ldots, \phi_j)$ and $(\rho\ \phi_1, \phi_2, \ldots, \phi_j)$ is adopted.

A flexible proposition can easily be described by a *fuzzy relation* [27], R, defined by a membership function $\mu_R(.) : \Phi_1 \times \Phi_2 \times \cdots \times \Phi_j \to K$, where $\Phi_1 \times \Phi_2 \times \cdots \times \Phi_j$ is the Cartesian product of the subsets of Φ allowable at this place in the proposition. A flexible operator, $o \in \mathrm{O}$ (Fig. 27(a)) is described by a fuzzy relation mapping from the precondition space onto a totally ordered *satisfaction scale*, L and a set of flexible effect propositions.

a)
(**operator** o
(**params** $param_1, param_2, \ldots$)
σ_i: {**when** (**preconds** $\theta_{i1}\ \theta_{i2} \ldots$)
(**effects** $\rho_{i1}\ \rho_{i2} \ldots$) l_i}
σ_j: {**when** (**preconds** $\theta_{j1}\ \theta_{j2} \ldots$)
(**effects** $\rho_{j1}\ \rho_{j2} \ldots$) l_j}

b)
(**goal** γ
{**when** $\theta_i\ l_i$)}
{**when** $\theta_j\ l_j$)}
etc)

Fig. 27. General formats of flexible operators and goals.

L is distinct from K in that it represents the degree of compromise required to apply an operator in the current situation as opposed to the truth of a proposition. As per K, L is composed of a finite number of membership degrees, $l_\perp, l_1, \ldots, l_\top$. The endpoints, $l_\perp \in L$ and $l_\top \in L$ respectively denote a complete lack of satisfaction and complete satisfaction.

A flexible operator consists of a set of disjoint conditional clauses, Σ. Each $\sigma \in \Sigma$ is a triple $\langle \Theta, \mathrm{E}, k_i \rangle$ denoting a conjunction of flexible preconditions, a conjunction of flexible effect propositions and the satisfaction degree of this operator given these preconditions. Each $\theta \in \Theta$ has the form $(\rho\ \phi_1, \phi_2, \ldots, \phi_j\ \tau\ \kappa)$, where τ is a precondition operator with argument set κ. Allowed precondition operators encompass equality, inequality, ranges of truth degrees and sets of discrete truth degrees. Each σ_i maps a subset of the space of preconditions to a set of effects and a satisfaction degree in L. Again, when dealing with preconditions which only ever take a truth value of $k_\perp$ or $k_\top$, the Boolean style of $\neg(\rho\ \phi_1, \phi_2, \ldots, \phi_j)$ and $(\rho\ \phi_1, \phi_2, \ldots, \phi_j)$ is adopted for θ.

A flexible goal $\gamma \in \Gamma$ maps from the space of flexible propositions to L. Each goal is defined using a number of clauses, as shown in Fig. 27(b). Preconditions are defined exactly as those used in the flexible operators. More than one set of mutually-consistent propositions may satisfy the plan goals to some extent. Hence, the satisfaction degree of a plan is the fuzzy conjunction of the satisfaction degrees of each operator and each goal used in the plan. A plan's *quality* is its satisfaction degree combined with its length, where the shorter of two plans with equivalent satisfaction degrees is better. A significant advantage of this formalism is in the over-constrained case: when Boolean planning returns no plan at all, compromise plans may still be constructed.

The disadvantage of using an idempotent operator is the so-called *drowning effect* [30], where a low satisfaction degree resulting from one assignment 'drowns' several others whose satisfaction degrees, whether optimal or sub-optimal, are not reflected in the overall satisfaction degree. This effect can be counteracted to an extent by exploring the path of highest satisfaction first, as demonstrated in the following section.

Fig. 28 presents an illustrative example, the Valuable Package Problem. Here, $K = \{k_\perp, k_1, k_2, k_\top\}$, $L = \{l_\perp, l_1, l_2, l_\top\}$. The goals (Fig. 29) are to transport the two packages to city c_3. The two packages have different values, described by: (*valuable* $pkg_1\ k_2$) and (*valuable* $pkg_2\ k_1$). Since pkg_2 is of a lower value, one option is to choose not to transport it, but at a very low satisfaction degree. Transportation is by means of a truck, initially

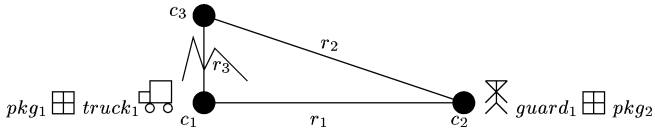

Fig. 28. A flexible planning example. The r_i are roads and the c_i are cities.

(**goal** pkg_1-destination (**when** (*at* $pkg_1\ c_3$) $l_\top$))	(**goal** pkg_2-destination (**when** (*at* $pkg_2\ c_2$) l_1) (**when** (*at* $pkg_2\ c_3$) $l_\top$))

Fig. 29. Flexible goals for the valuable package problem.

a) (**operator** `Drive-truck`
params (*?v* `vehicle`) (*?o* `location`) (*?d* `location`) (*?rMj* `mj-rd`) (*?rMt* `mtn-rd`))
{**when** (**preconds** (*at ?v ?o*) (*connects ?rMj ?o ?d*))
(**effects** (*not* (*at ?v ?o*)) (*at ?v ?d*)) $l_\top$}
{**when** (**preconds** (*at ?v ?o*) (*connects ?rMt ?o ?d*))
(**effects** (*not* (*at ?v ?o*)) (*at ?v ?d*)) l_1})

b) (**operator** `Load-truck`
params (*?t* `truck`) (*?p* `package`) (*?d* `location`) (*?g* `guard`))
{**when** (**preconds** (*at ?t ?l*) (*at ?p ?l*) (*valuable ?p* $<= k_1$))
(**effects** (*not* (*at ?p ?l*)) (*on ?p ?t*)) $l_\top$}
{**when** (**preconds** (*at ?t ?l*) (*at ?p ?l*) (*on ?g ?t*) (*valuable ?p* $>= k_2$))
(**effects** (*not* (*at ?p ?l*)) (*on ?p ?t*)) $l_\top$}
{**when** (**preconds** (*at ?t ?l*) (*at ?p ?l*) (*not* (*on ?g ?t*)) (*valuable ?p* $>= k_2$))
(**effects** (*not* (*at ?p ?l*)) (*on ?p ?t*)) l_2})

Fig. 30. The `Drive-truck` and `Load-truck` operators.

at c_1. Road r_3 is mountainous and should be avoided if possible, while the remainder are main roads. The flexible operator `Drive-truck` (Fig. 30(a)) describes how the truck moves across the map. Fig. 30(b) presents `Load-truck` which considers the value of the package and the presence of a guard. If the package is not very valuable (e.g., pkg_2), the guard's presence is immaterial—a satisfaction degree of $l_\top$ is assigned. If the package is quite valuable, however, loading it onto the truck without a guard results in a satisfaction degree of l_2. A similar flexible operator, `Unload-truck`, also exists. Further (Boolean) operators allow the guard to board and leave the truck.

6. Flexible Graphplan: synthesising plans under soft constraints

Similarly to Graphplan, Flexible Graphplan (FGP) constructs and analyses Flexible Planning Graphs in two interleaved phases: a forward phase where the graph is extended until the plan goals are found, and a backward phase where the graph is searched for a plan. A planning graph is divided into a number of *levels*. $Level_i$ contains actions ($actions_i$) and propositions ($propositions_i$). $Level_0$ contains proposition nodes which capture the initial problem state.

Plan synthesis can be simplified greatly by adding mutual exclusion constraints to the planning graph. Two propositions are exclusive if they express different truth degrees for the same proposition or all ways of creating one are exclusive of all ways of creating the other. Exclusion constraints between action nodes express that no valid plan contains both actions. Two actions are mutually exclusive for three reasons, as per Graphplan: *Inconsistent Effects*—an effect of one action expresses a different truth degree from an effect of the other for the same proposition; *Interference*—an effect of one action expresses a different truth degree from a precondition of the other for the same proposition; *Competing Needs*—the actions have mutually exclusive preconditions.

When extending the planning graph to $level_i$ from $level_{i-1}$ (Fig. 31), each clause of each operator is instantiated in all possible ways to mutually consistent nodes in $propositions_{i-1}$.

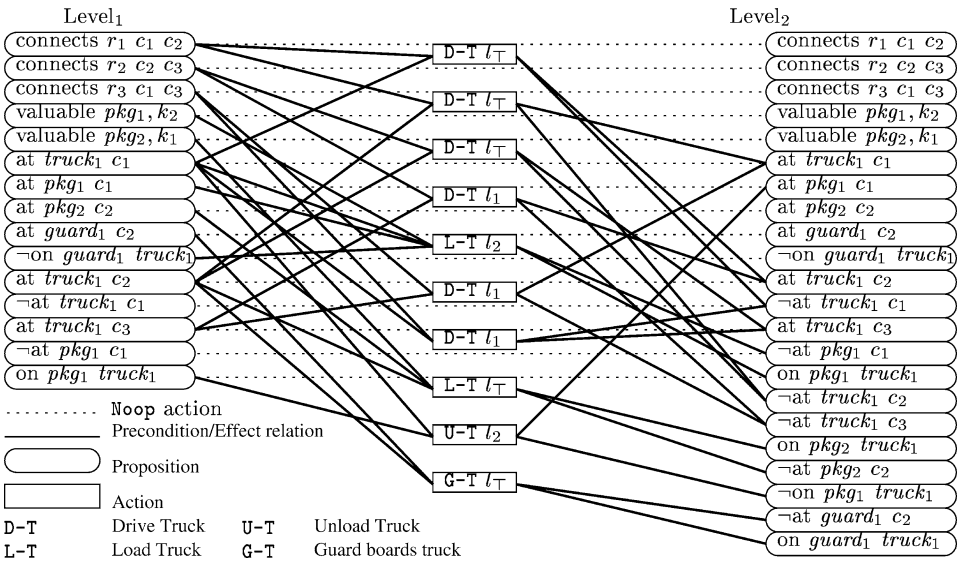

Fig. 31. Flexible planning graph $levels_{1,2}$ for the problem of Fig. 28. All pairs of non-Noop nodes in $actions_2$ except {G-T $l_\top$, L-T $l_\top$} are mutually exclusive.

An action with the satisfaction degree for this instantiation is added to $actions_i$. A Noop records the persistence of a proposition. Mutual exclusion relations are added according to the above conditions.

6.1. Limited graph expansion and satisfaction propagation

Flexible graph expansion has an increased cost compared with Boolean graph expansion, though reducible by imposing limits on graph expansion. If a plan of satisfaction degree $l_\alpha < l_\top$ has been found, FGP searches onwards to look for a plan with a higher satisfaction degree. Yet, there is no point in instantiating flexible operator clauses with a satisfaction degree less than or equal to l_α: a plan with this satisfaction degree has been found already—a longer plan with the same satisfaction degree is deemed to be of a lower quality. The conjunctive combination rule implemented via the *min* operator ensures that no plan of satisfaction degree l_b can contain an action of satisfaction l_a, where $l_a < l_b$. Hence, the completeness of the search is not affected by omitting such actions in further graph levels.

Propagating satisfaction degrees forwards as the graph is expanded can also considerably reduce search effort. The conjunctive combination rule guarantees that the overall satisfaction degree of an action plus its supported preconditions is the minimum satisfaction degree over all the actions involved. Hence, while expanding the graph to $level_i$, keep track of the maximum satisfaction degree of all actions that produce each node in $propositions_i$. When expanding the graph to $level_{i+1}$ the satisfaction degree of a node in $actions_{i+1}$ is the minimum of the satisfaction degree of the instantiated operator clause and the minimum of maximum satisfaction degrees of each of its preconditions. Search is

reduced because it is obvious at an earlier stage that, say, a certain action can never be part of a plan with satisfaction degree $l_\top$. For instance, `Unload-truck` (Fig. 31) requires 'on pkg_1 $truck_1$' as a precondition. The sole way of asserting this proposition at $level_1$ is via a `Load-truck` action which has a satisfaction degree of l_2. The other precondition of `Unload-truck`, 'at $truck_1$ c_1', is an initial condition, so the minimum of maximum satisfaction degrees attached to the preconditions of this action is l_2. The satisfaction degree of `Unload-truck` at $level_2$ is the minimum of l_2 and the satisfaction degree of the instantiated operator clause ($l_\top$), resulting in a satisfaction degree of l_2.

6.2. Flexible plan extraction via fuzzy rrDFCSP

A node in $propositions_i$ can be viewed as a fuzzy CSP variable, its domain comprising the set of nodes in $actions_i$ which assert this proposition as an effect. A unary *preference* constraint is constructed from the domain elements and their associated satisfaction degrees. Consider 'at $truck_1$ c_1' in $propositions_2$ (Fig. 31) with a domain of two `Drive-truck` actions and a `Noop` action. A unary preference constraint specifies that the assignment of one of the `Drive-truck` actions and that of the `Noop` action have satisfaction degree $l_\top$, while the assignment of the other `Drive-truck` action has a satisfaction degree of l_1. Boolean binary constraints are generated directly from the mutual-exclusion relations in the planning graph.

If $level_g$ contains the goal propositions, each set of supporting actions specifies (via preconditions) a sub-problem at $level_{g-1}$. Solutions to these sub-problems specify new sub-problems at $level_{g-2}$, etc. A sub-problem sequence associated with a graph level is viewed as a fuzzy *rr*DFCSP (Fig. 32). FGP uses ACFLCFC to solve these sequences since individual sub-problems are relatively easy to solve, and ACFLCFC is relatively lightweight. This algorithm is also able to maintain solution stability. A stable solution to a sequence of sub-problems enables FGP to focus on conflicts that prevent the subgoal propositions under consideration from being supportable.

If solutions to the sub-problem at $level_i$ intersect (likely in all but the most tightly constrained cases), there will be similarities between sub-problems at $level_{i-1}$. An *rr*DFCSP algorithm exploits these similarities, re-using effort applied to the previous problem in the sequence when solving the current sub-problem. Effort used in levels visited in a failed plan extraction is also re-used when the graph is expanded and a further attempt is made.

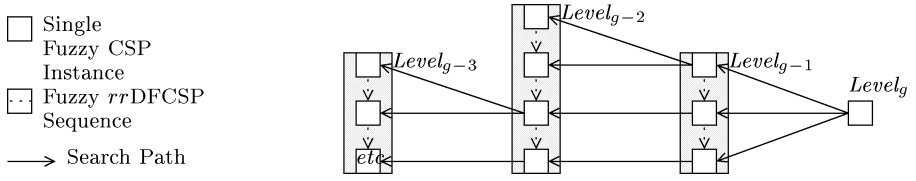

Fig. 32. Plan extraction as a hierarchy of fuzzy *rr*DCSPs.

6.3. Memoisation

To increase efficiency in searching for a valid plan, branches of the search which necessarily lead to failure must be pruned early. A mutually-consistent set of nodes in $propositions_i$ is *conjunctively unsupportable* if all combinations of nodes in $actions_i$ that assert these propositions are either directly inconsistent or their own preconditions are unsupportable. *Memoisation* is the process of recording such unsupportable sets, known as *memosets* [24]. If a memoset (or its superset) is encountered again at the level at which it was recorded, the current search branch can be pruned immediately. Naively, the entire set of nodes currently considered at $propositions_i$ may be recorded if it is established that they are unsupportable. However, it is likely in general that certain propositions are irrelevant to the failure. More effective memoisation can significantly improve plan extraction efficiency.

Memoisation is implemented via enforcing fuzzy arc consistency. A *reduction explanation* for l_a of x_r for domain element d_i records the fact that x_r is responsible for the revision of the unary consistency of d_i below l_a. This situation occurs when there are no elements in D_r that are compatible with d_i and have a satisfaction degree of at least l_a. Figs. 33(a) and (b) show a sub-problem before and after the revision of $arc(2,1)$ and $arc(2,3)$. Consider the first two elements of D_2: both are compatible only with elements in D_1 with satisfaction degrees of at most l_2. Their satisfaction degrees can therefore be reduced to l_2. Reduction explanations record that x_1 is responsible for this reduction. Similarly, the consistency of the third element of D_2 is reduced to l_1 due to the constraint with x_3. Now the maximum satisfaction degree of any element in D_2 is l_2. Hence, the satisfaction degree of the problem is also reduced to l_2, since some element in D_2 *must* be assigned to x_2 in any solution.

Using the reduction explanations, a memoset is constructed containing the variables which conjunctively do not admit a solution with a particular satisfaction degree. This is done by tracing recursively the reduction explanations as detailed in [23]. Enforcing fuzzy arc consistency is not guaranteed to detect that a problem does not contain a solution of satisfaction degree l_a. A more complex inconsistency may only be uncovered during subsequent search. The current search branch cannot lead to a solution of satisfaction degree l_a when all elements of D_i that are compatible with assignments to existing variables have a unary satisfaction degree below l_a. In this case, the currently assigned

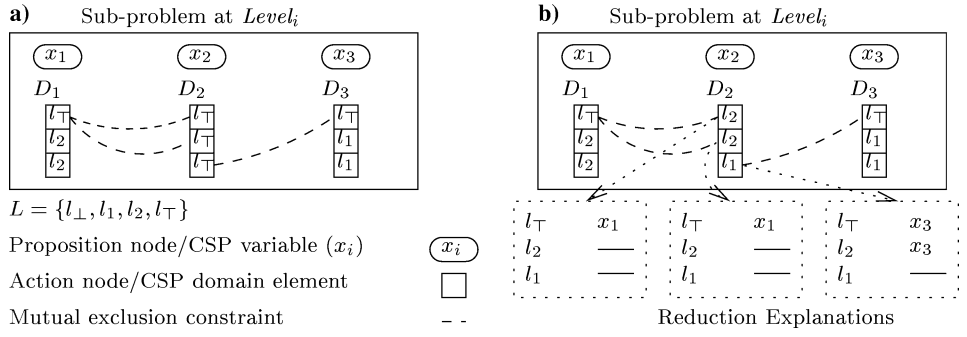

Fig. 33. Constructing memosets by tracing reduction explanations.

variables form a conflict set with x_i with respect to l_a. This inconsistency can be recorded as a partial memoset for the satisfaction degree l_a. Since flexibility is present only in the unary preference constraints associated with each variable, variables with assignments having a satisfaction degree of l_a or above are irrelevant to this conflict and removed. If no solution exists with satisfaction degree l_a, an l_a-memoset is constructed from the union of the recorded partial memosets.

Memoset information from child sub-problems is used to guide further solution extraction at the parent sub-problem. Memosets from a sub-problem at $level_{i-1}$ are mapped back onto action nodes in a solution assignment at $level_i$, which generated the sub-problem at $level_{i-1}$, using precondition relations in the planning graph. Sub-assignments thus specified represent new constraints that can be added to the current instance of a sub-problem at $level_i$, creating a new instance in the dynamic sequence. This process is *not* the same as conflict recording during the solution of a single CSP, since the new constraints come from an external source. FLC solves the new instance by repairing the previous solution. Propagation means that FLC only has to discover one solution for each of a limited sequence of increasingly constrained sub-problems, rather than attempting to enumerate all solutions to a single sub-problem. This is more efficient since each propagated repair can potentially exclude many solutions to the original sub-problem. The propagated repair process emphasises the usefulness of *rr*DFCSP techniques in this hierarchical context.

6.4. The FGP algorithm

FGP is presented in pseudo-code form in Fig. 34. For reasons of space, the proof of the soundness and completeness of FGP is omitted (see [23]). FGP can solve Boolean problems by restricting the truth and satisfaction scales to $\{k_\bot, k_\top\}$ and $\{l_\bot, l_\top\}$ respectively. The procedure `FGP( )` is the top level of the algorithm. The planning graph (*graph*), the current ($plan_\beta$) and best known ($plan_\alpha$) plans have global scope. Lines 4–9 extend the graph and search for plans until a compromise-free plan is encountered. At each graph level, the *goalSets* of proposition nodes which are mutually-consistent and match the plan goals are determined (line 6). Since flexible goals associate a satisfaction degree with different propositions, a combined satisfaction degree is generated for each *goalSet*. The *goalSets* are sorted in descending order of satisfaction degree (line 7) before attempting to find a plan containing them (line 9). Sorting increases the likelihood of finding a good plan early, which becomes a bound for future search.

The `extract( )` and `extractLevel( )` procedures control the solution of the hierarchy of fuzzy *rr*DFCSPs created from the planning graph. The former simply initialises a partial plan with the satisfaction degree of the current *goalSet* (line 2) before calling the latter. The base case of this recursion is when $level_0$ of the planning graph is reached (lines 2–4). If the subgoals now match the initial conditions (line 2), a new best plan has been found and becomes the bound against which future partial plans are compared (line 3).

At all other levels, FGP first checks whether a previously recorded memoset shows that the current branch cannot lead to a new best plan. If so, this branch of the search is pruned, and the memoset is returned to the level above for propagation (line 6). Otherwise, the instance of `BBFCLex` dealing with this level of the graph is retrieved (line 7) and a

```
1.  Procedure FGP(ψ)
2.    graph ← graph-initialise(ψ)
3.    level ← 0, l_α ← l_⊤
4.    While Not compromise-free(plan_α)
5.      level ← level+1
5.      extend(graph)
6.      goalSets ← getGoalSets (graph, ψ)
7.      sort(goalSets)
8.      Foreach goalSet ∈ goalSets
9.        If (satDegree(goalSet) > l_α) extract(level, goalSet)
1.  Procedure extract(level, goalSet)
2.    plan_β ← plan-initialise(level, satDegree(goalSet))
3.    extractLevel(level, getPropositionNodes(goalSet))
1.  Procedure extractLevel(level, subGoals)
2.    If (level = 0 And matchInitConds(subGoals))
3.      plan_α ← plan_β
4.      Return ∅
5.    memoset ← getMemoset(graph, level, subGoals)
6.    If (memoset ≠ ∅) Return memoset
7.    ACFLCFC_l ← getSolver(graph, level)
8.    subProb ← translate(subGoals)
9.    memoset ← solve(ACFLCFC_l, subProb)
10.   If (memoset ≠ ∅) recordMemo(memoset, level)
11.   Return memoset
```

Fig. 34. Overview of FGP.

new sub-problem in the corresponding *rr*DFCSP is generated (line 8). This sub-problem is solved, returning a new memoset to be recorded and propagated (lines 9–11). `ACFLCFC()` and `extractLevel()` are mutually recursive: once the former has found a solution, it invokes the latter to search the next level down.

6.5. Complexity issues of FGP

The flexibility inherent in the flexible planning graph contributes to the increased complexity of FGP over Graphplan. As ℓ, the number of satisfaction degrees in the scale L increases, there is an increase in the number of sub-problems that FLC must solve. This is because solutions of all satisfaction degrees above $l_\perp$ must be explored. This increase is, however, in the average rather than the worst case, since with a constant number of sub-problem variables and domain size (neither of which are affected by ℓ) the maximum number of potential solutions to explore remains the same.

The time complexity of FLC has been established as $O(nm^n)$ [23]. If the number of plan goals in a g-level planning graph is G, m is again the domain size (i.e., the maximum number of action nodes that assert a particular proposition) and p is the maximum number of preconditions for any of the operators in the problem, the worst-case time complexity of synthesising a plan can be expressed by (see [23] for proof):

$$\sum_{i=1}^{g} Gp^{i-1} m^{2Gp^{i-1}} \prod_{j=2}^{i} m^{Gp^{j-2}} \approx g\left[Gp^{g-1} m^{2Gp^{g-1}} m^{Gp^{g-1}}\right].$$

That is,

$$\mathrm{O}\left(gGp^{g-1} m^{3Gp^{g-1}}\right).$$

If there are G flexible goals, the worst case occurs when each can be satisfied in all $\ell - 1$, producing $G^{\ell-1}$ goal combinations. Hence, the worst case time complexity of plan synthesis from $level_g$ of a flexible planning graph is:

$$\mathrm{O}\left(gG^{\ell} p^{g-1} m^{3Gp^{g-1}}\right).$$

The worst-case space required by a flexible planning graph is no greater than that required by a Boolean planning graph. Reconsider the space complexity analysis made in [3] of a Boolean planning problem with n objects, i propositions in the initial conditions, s STRIPS operators each with k parameters and an effects list of size e. Operators cannot create new objects, so the number of different propositions that can be created by instantiating an operator is $\mathrm{O}(en^k)$. Hence, the maximum number of nodes in any proposition level is $\mathrm{O}(i + sen^k)$. Since any operator can be instantiated in at most $\mathrm{O}(n^k)$ different ways, the maximum number of nodes in any action level of the graph is $\mathrm{O}(en^k)$. Flexible operators have several clauses, but these clauses must be disjoint (Section 5). Hence, the number of ways any operator can be instantiated remains as $\mathrm{O}(n^k)$. Including memoisation, FGP has exponential worst case space complexity because an exponential number of sub-problems is solved in the worst case, each of which can lead to the recording of a memoset. This result reinforces the need for informed memoisation.

6.6. *Synthesised plans for the valuable package problem*

The first short plan synthesised by FGP is as follows, the major compromises being the route across the unsafe track and the fact that only pkg_1 is delivered.

(1) `Load-truck` pkg_1 $truck_1$ (l_2)
(2) `Drive-truck` $truck_1$ c_1 to c_3 via r_3 (l_1)
(3) `Unload-truck` pkg_1 $truck_1$ ($l_\top$)

Using main roads, both packages can be carried to c_3 in 5 steps, with a satisfaction degree of l_2 since the guard is not present when pkg_1 is loaded.

(1) `Load-truck` pkg_1 $truck_1$ (l_2)
(2) `Drive-truck` $truck_1$ c_1 to c_2 via r_1 ($l_\top$)
(3) `Load-truck` pkg_2 $truck_1$ ($l_\top$)
(4) `Drive-truck` $truck_1$ c_2 to c_3 via r_2 ($l_\top$)
(5) • `Unload-truck` pkg_1 $truck_1$ ($l_\top$)
    • `Unload-truck` pkg_2 $truck_1$ ($l_\top$)

The above plan is still not regarded as having the highest satisfaction degree because the guard was not present when the first package was loaded. The 'no compromise' plan contains seven steps and is shown below.

(1) `Drive-truck` $truck_1$ c_1 to c_2 via r_1 $(l_\top)$
(2) • `Guard-gets-on-truck` $truck_1$ $guard_1$ $(l_\top)$
    • `Load-truck` pkg_2 $truck_1$ $(l_\top)$
(3) `Drive-truck` $truck_1$ c_2 to c_1 via r_1 $(l_\top)$
(4) `Load-truck` $truck_1$ $package_1$ $guard_1$ $(l_\top)$
(5) `Drive-truck` $truck_1$ c_1 to c_2 via r_1 $(l_\top)$
(6) `Drive-truck` $truck_1$ c_2 to c_3 via r_2 $(l_\top)$
(7) • `Unload-truck` pkg_1 $truck_1$ $(l_\top)$
    • `Unload-truck` pkg_2 $truck_1$ $(l_\top)$

This simple example demonstrates how a range of plans of different lengths and containing alternative compromises can be synthesised from a given flexible planning problem. The user can then select the one which is deemed to offer the best compromise between length and satisfaction degree.

7. FGP: experimental results

A test suite of twelve problems containing plans of three satisfaction degrees at different distributions of plan length is used to examine the relationship between solution distribution and search effort. Boolean versions of the twelve problems are also solved to investigate the overhead incurred by searching for a range of plans. The utility of limited graph expansion and satisfaction propagation (Section 6.1) is also examined. The Rescue problem [24] is then described. It exhibits complex interactions between flexible operators and goals, testing the ability of FGP to trade plan length versus the compromises made. For reasons of space, results on Boolean benchmarks are omitted here, but can be found in [24].

7.1. The test suite

The test suite contains logistics problems of the same type as the example in Section 5: the target is to transport packages to their respective destinations. To compare the effort of searching for a range of solutions with synthesising a single plan from a Boolean version of the same problem, two sets of operators are used. The first are flexible and allow compromises to be made to synthesise one or more shorter plans as well as a plan containing no compromises. Let $K = \{k_\bot, k_1, k_2, k_\top\}$ and $L = \{l_\bot, l_1, l_2, l_\top\}$. The 12 problems have been constructed such that it is always possible to synthesise three different plans (Fig. 35). It is preferred that the guard is present when a package is loaded onto a truck (if not, the satisfaction degree of `Load-truck` is l_2). It is also preferred not to transport valuable packages through dangerous areas (otherwise the satisfaction degree of `Drive-truck` is l_1). A second set of Boolean operators make imperative the

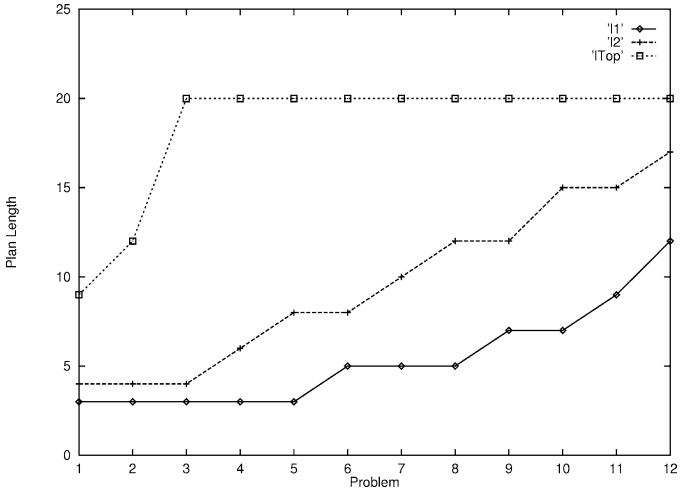

Fig. 35. Test suite: plan lengths per satisfaction degree.

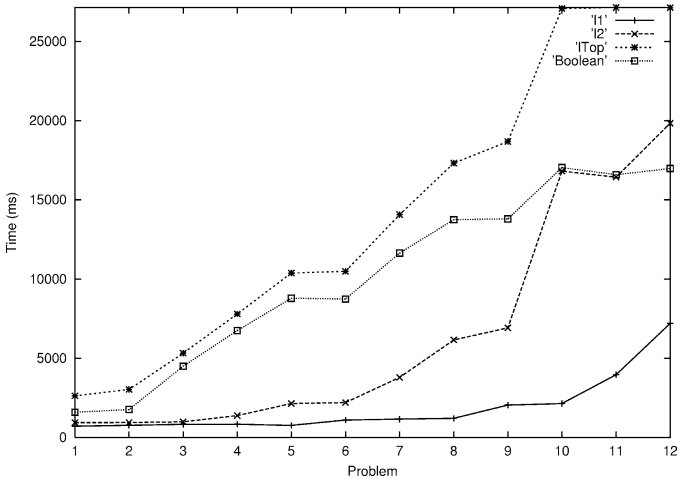

Fig. 36. Test suite: flexible and Boolean operators.

preferences in `Load-truck` and `Drive-truck`, using the endpoints of K and L. This creates a Boolean version of each problem, allowing the synthesis of the shortest plan with satisfaction degree $l_\top$ only.

The 12 problems are solved using FGP and both flexible and Boolean operators (Fig. 36). As expected, it always takes longer to find a compromise-free plan when also searching for shorter compromise plans than to solve the Boolean problem. However, the time taken to produce compromise plans is significantly lower than to solve the Boolean problem, providing ‘anytime’ behaviour.

The difficulty of Boolean plan synthesis tracks the use of the flexible operators to find the $l_\top$ plan (Fig. 36). Although problems 3–12 all have the same length of compromise-free plan, the increasing effort required to solve them is reflected in the increasing difficulty of making compromises to find shorter plans using the flexible operators. When a plan with satisfaction degree l_i has been found, FGP no longer needs to search for plans nor consider actions of satisfaction degree less or equal to l_i. This explains why the earlier problems in the test suite, where the l_1 and l_2 plans have short lengths, are easiest to solve. In problems 10, 11 and 12 FGP spends a lot of time looking for plans of all three non-$l_\perp$ satisfaction degrees before one of l_1 is discovered.

The difficulty of synthesising the l_1 plan closely corresponds to its length. All propositions are asserted by an action with at least satisfaction degree l_1, hence all actions in the planning graph can be considered for inclusion in an l_1 plan, resulting in a less complex search process. The difference in length between the l_1 and l_2 plans seems to be most influential on the amount of overall search effort (e.g., problem 10). The region of the search where only an l_1 plan has been found is likely to be the most intensive. Prior to the discovery of the l_1 plan, a significant number of levels of the graph can be constructed without any search being required since a mutually-consistent set of propositions matching the plan goals do not yet exist. After the l_2 plan has been found, the search becomes easier, discarding all actions with a satisfaction degree below $l_\top$.

7.2. The utility of limited graph expansion and satisfaction propagation

To gauge the efficacy of limited graph expansion and satisfaction propagation (Section 6.1) results were first obtained on the test suite with a version of FGP which uses neither technique (Fig. 37). A clear deterioration is evident compared with times in Fig. 36. Time to synthesise the l_1 plan is, however, unaffected. Limited graph expansion has no effect until at least the l_1 plan is found. Since l_1 is the lowest non-$l_\perp$ satisfaction degree,

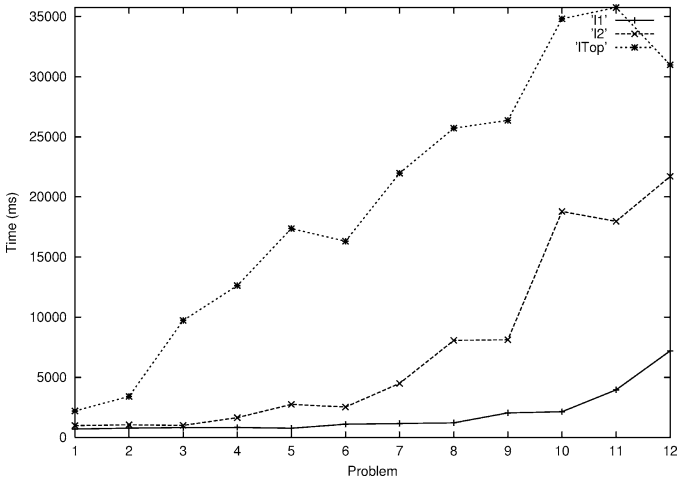

Fig. 37. Test suite: no limited graph expansion or satisfaction propagation.

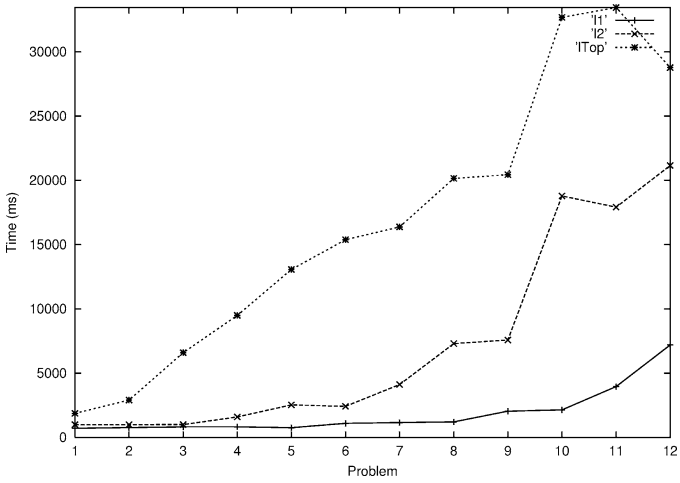

Fig. 38. Test suite: flexible operators and limited graph expansion only.

satisfaction propagation cannot revise the satisfaction degree of any action below this degree.

Effort required to synthesise the l_2 plans is increased, but not to the same extent as that required for the $l_\top$ plans. The utility of limited graph expansion increases with the number of satisfaction degrees it can rule out. Without limited graph expansion, actions of all satisfaction degrees are continually added to the graph, increasing its size and the cost of processing it into sub-problems. Satisfaction propagation also provides most benefit to search for plans with the highest satisfaction degrees. Revision of action satisfaction degrees is down L; without satisfaction propagation, many more actions have higher satisfaction degrees associated, requiring more search to exclude them.

To determine which technique has the most beneficial effect on the plan synthesis process, the test suite problems were solved twice more, using only limited graph expansion or satisfaction propagation in each case (Figs. 38, 39). On this test suite, satisfaction propagation is the more effective technique. This is expected to be generally true: limited graph expansion decreases the size of the flexible planning graph and the time taken both to expand it and to construct sub-problems, whereas satisfaction propagation improves the efficiency of plan extraction, the dominant cost of plan synthesis.

7.3. The rescue problem

To demonstrate the utility of flexible planning on a more substantial problem, the example given in Fig. 40 is used. The scenario is a rescue operation: a volatile volcano on Volcano Island has started to erupt. The target is to evacuate scientists and civilians present on the island and *preferably* their equipment and belongings. There are three points to make the evacuation to, with ascending degrees of satisfaction: the (relatively) 'Safe Point' on the other side of Volcano Island, the nearby village on Neighbour Island and finally the safety of Far Island. A minimum target is to evacuate the people to Safe Point. Slightly

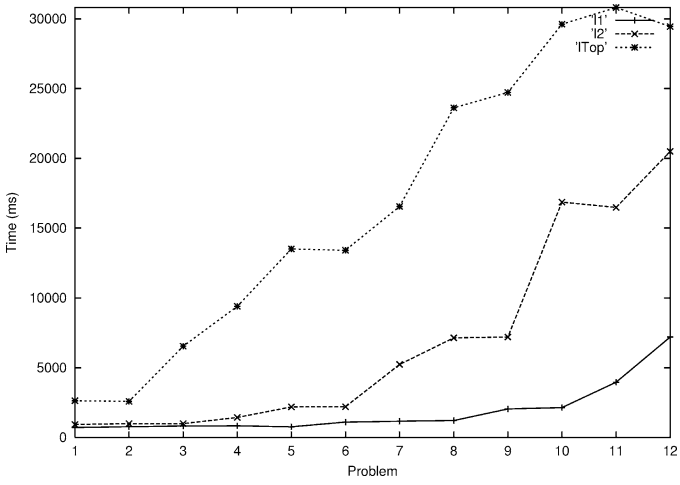

Fig. 39. Test suite: flexible operators and satisfaction propagation only.

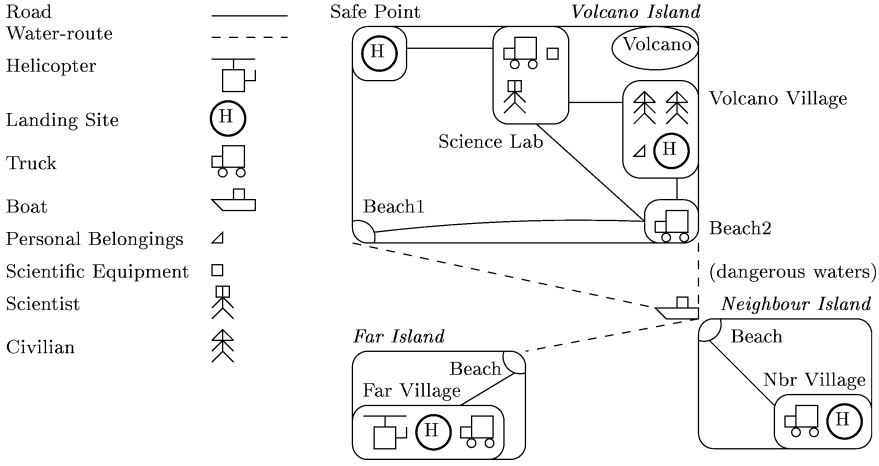

Fig. 40. Overview of the rescue problem.

better is to save also the equipment and possessions. A target with a higher satisfaction is to remove people and possessions to Neighbour Village. The ‘no compromise’ target is to evacuate all people to Far Island and make sure that all equipment and possessions are removed from Volcano Island.

There are several other constraints and sources of flexibility which lead to a range of different plans. Firstly, the helicopter is small and usually carries only one passenger. Because of the emergency it is possible to get two passengers on board, although this is not a preferred option. There is no facility to carry equipment/belongings in this vehicle. The pilot would also like to avoid Volcano Island, if at all possible. The trucks and boat

can carry people or possessions. One of the routes that the boat may take between Volcano Island and Neighbour Island is treacherous, and to be avoided if possible.

This problem uses the satisfaction scale $L = \{l_\bot, l_1, l_2, l_3, l_4, l_\top\}$ and a truth degree scale, K, of the same size. This is not a trivial problem: there are 432 possible flexible goal combinations alone. FGP synthesises 5 plans for the rescue problem in under 30 seconds, one per satisfaction degree, discounting $l_\bot$. This is not the case for all problems since, as noted, given two plans of the same length the one with the higher satisfaction degree is returned. There are, of course, alternative plans with the same satisfaction degree and length, but this is no different from the situation found in Boolean planning.

The shortest plan of four steps is shown below. Although other actions in the plan are 'drowned' by the fact that the helicopter must fly to Volcano Village, by exploring goals with highest satisfaction degree first this has been counteracted: the helicopter is flown to Far Village when a plan with an equivalent satisfaction degree would be to take the civilians to the Safe Point.

(1) `LOAD-TRUCK` lab-truck scientific-equipment ($l_\top$)
    `LOAD-TRUCK` lab-truck scientist ($l_\top$)
    `FLY-HELICOPTER` Far Village, Volcano Village (l_1)
(2) `LOAD-HELICOPTER` civilian1, civilian2 (l_4)
    `DRIVE-TRUCK` lab-truck Science-lab, Safe Point ($l_\top$)
(3) `UNLOAD-TRUCK` lab-truck scientific equipment ($l_\top$)
    `UNLOAD-TRUCK` lab-truck scientist ($l_\top$)
    `FLY-HELICOPTER` Volcano Village, Far Village ($l_\top$)
(4) `UNLOAD-HELICOPTER` civilian1, civilian2 ($l_\top$)

A 5-step plan (satisfaction degree l_2) transports people and possessions to Safe Point. The major compromise is that the people remain on Volcano island. A 9-step plan (satisfaction l_3) evacuates the people from Volcano Island in the least number of steps. The boat takes the scientist and equipment to Neighbour Island, while a helicopter rescues both civilians from Safe Point. Drowning is avoided: for example, the civilians are flown to Far Village, when flying them to Neighbour Village produces a plan with the same satisfaction degree. An 11-step (satisfaction l_4) plan compromises only in the destination of the people: the boat evacuates the people, equipment and possessions (avoiding dangerous waters) to Neighbour Village. A 'no compromise' plan takes 12 steps. The helicopter avoids volcano island and the boat avoids the dangerous route between Volcano Beach2 and Neighbour Beach. The flexible goals are also completely satisfied, with all people, equipment and possessions safely at Far Village. Each plan is returned as soon as it is found, providing anytime solutions. Each offers a different balance between compromises made and plan length. A user can select the plan considered to make the optimal tradeoff.

8. Conclusion

This paper has been concerned primarily with fuzzy dynamic flexible constraint satisfaction problems (fuzzy *rr*DFCSPs). These problems represent the integration of

two previously separate extensions to classical constraint satisfaction which address two of its main limitations: the inability either to cope gracefully with changes to the problem structure or to support compromise in an over-constrained problem. The structure and properties of fuzzy *rr*DFCSP were investigated using a range of random problem sequences and two types of solution procedure. The first extends a branch and bound (BB) approach to static fuzzy CSP [9] to cope with dynamic changes, the second (FLC) extends the Local Changes [35] Boolean DCSP approach to support fuzzy constraints.

Results from solving these sequences showed a number of peaks in solution difficulty corresponding to multiple phase transitions caused by a number of degrees of consistency above $l_\top$. This generalises the phenomenon of a single phase transition found in empirical studies of classical CSP. Around the hardness peaks, the enhanced BB algorithm found solutions most efficiently, due to its relatively more effective constraint propagation. FLC variants consistently produced more stable solutions, which is important if significant effort has been invested in the execution of the current solution.

Fuzzy *rr*DFCSP was applied to AI Planning. The *flexible planning problem* is founded on the use of subjective truth and satisfaction degree scales. Truth degrees associated with propositions support uncertainty concerning the exact state of the environment. Satisfaction degrees associated with operators and goals enable the planner to make compromises, assigning an appropriate degree of satisfaction to each instantiated operator and goal according to how well their preconditions are satisfied. Flexible plans trade plan length versus compromises made, supporting an anytime behaviour: given limited time, the best plan found so far can be returned. To synthesise plans from an input flexible planning problem, Graphplan was extended to create the Flexible Graphplan (FGP). FGP hierarchically decomposes the planning graph, creating a fuzzy *rr*DFCSP from each graph level which is solved with FLC.

FGP was tested on a suite of problems containing plans of certain satisfaction degrees of particular lengths, and also on a more complex benchmark problem. Performance was sensitive to internal structure. If a short 'compromise' plan can be found, FGP can concentrate on finding plans with fewer compromises for the remainder of the search, reducing effort. Unsurprisingly, it is more expensive to look for a range of plans than to search for one compromise-free plan. However, it is often possible to find short, low satisfaction plans quickly, supporting the anytime behaviour. The user is provided with a range of options trading plan length versus the number and severity of the compromises made.

A principal element of future work is the development of a greater number of fuzzy *rr*DFCSP solution techniques. Further solution techniques could be created through the extension of alternative existing dynamic or flexible algorithms. For example, a local search based algorithm could be used, performing hill climbing on the quality of the current solution as dictated by the fuzzy constraints. As has been shown for purely dynamic CSP [37], hill-climbing can be especially useful in rapidly evolving problems, since it is not reliant on the stability of previously assigned variables. This point is equally valid for the dynamic flexible case.

The flexible planning problem uses idempotent min/max aggregation operator to calculate satisfaction degrees, allowing the straightforward application of efficient classical CSP techniques. The disadvantage is the drowning effect, as noted in Section 5. Leximin

fuzzy CSP [26] is one possible solution. This approach combines additive and min/max operations to consider the satisfaction of all constraints, but has a higher computational cost. Alternatively, additive aggregation could be employed. This supports a finer-grained form of flexibility, and avoids the drowning effect. Recently developed means of enforcing arc consistency for non-idempotent operators could then be brought to bear [6,29]. As described in [23], FLC could easily be modified to support various types of flexible CSP, such as those covered by valued or semiring CSP [2], to support other instances of dynamic flexible CSP. The structure of the algorithm need not be changed, but the method of consistency degree aggregation must be considered when enforcing consistency and generating bounds for repair sub-problems. A similar empirical analysis to that made here could then be performed to establish the utility of FLC in each case.

Finally, apart from AI Planning, it is interesting to investigate the application of fuzzy *rr*DFCSP to many other AI problems. Work is, for instance, ongoing in using fuzzy *rr*DFCSP to the selection of the most preferred model in compositional modelling [16].

Acknowledgements

This work is partially supported by UK EPSRC grants 97305803 and GR/N16129. The authors thank Peter Jarvis for his assistance in this research and the anonymous referees for their insightful comments which were very useful in revising this paper.

References

[1] J. Allen, J. Hendler, A. Tate, Readings in Planning, Morgan Kaufmann, San Mateo, CA, 1990.
[2] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, G. Verfaillie, H. Fargier, Semiring-based csps and valued csps: Frameworks, properties, and comparison, Constraints 4 (3) (1999) 199–240.
[3] A. Blum, M. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1–2) (1997) 281–300.
[4] D. Brelaz, New methods to colour the vertices of a graph, J. ACM 22 (4) (1979) 251–256.
[5] S. Chien, Static and completion analysis for planning knowledge base development and verification, in: Proc. 3rd International Conference on Artificial Intelligence Planning Systems, Edinburgh, UK, 1996, pp. 53–61.
[6] M. Cooper, Reduction operations in fuzzy and valued constraint satisfaction, Fuzzy Sets and Systems 134 (2003) 311–342.
[7] R. Dechter, Constraint networks, in: Encyclopedia of Artificial Intelligence, Wiley, New York, 1992, pp. 276–285.
[8] R. Dechter, A. Dechter, Belief maintenance in dynamic constraint networks, in: Proc. AAAI-88, St. Paul, MN, 1988, pp. 37–42.
[9] D. Dubois, H. Fargier, H. Prade, Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty, Appl. Intelligence 6 (1996) 287–309.
[10] H. Fargier, Problèmes de satisfaction de constraintes flexibles: Application à l'ordonnancement de production, Ph.D. Thesis, Université Paul Sabatier, Toulouse, 1994.
[11] E. Freuder, Partial constraint satisfaction, in: Proc. IJCAI-89, Detroit, MI, 1989, pp. 278–283.
[12] I. Gent, E. MacIntyre, P. Prosser, P. Shaw, T. Walsh, The constrainedness of arc consistency, in: Proc. 3rd International Conference on Principles and Practice of Constraint Programming, Linz, Austria, 1997, pp. 327–340.
[13] I. Gent, E. MacIntyre, P. Prosser, B. Smith, T. Walsh, An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem, in: Proc. 2nd International Conference on Principles and Practice of Constraint Programming, Cambridge, MA, 1996, pp. 179–193.

[14] M. Ginsberg, Dynamic backtracking, J. Artificial Intelligence Res. 1 (1993) 25–46.
[15] R. Haralick, G. Elliot, Increasing tree search efficiency for constraint satisfaction problems, Artificial Intelligence 14 (1980) 263–313.
[16] J. Keppens, Q. Shen, On compositional modelling, Knowledge Engineering Rev. 16 (2) (2001) 157–200.
[17] A. Mackworth, Consistency in networks of relations, Artificial Intelligence 8 (1) (1977) 99–118.
[18] A. Mackworth, Constraint satisfaction problems, in: Encyclopedia of Artificial Intelligence, Wiley, New York, 1992, pp. 285–293.
[19] B. Mazure, L. Saïs, E. Grégoire, Tabu search for SAT, in: Proc. AAAI-97, Providence, RI, 1997, pp. 281–285.
[20] I. Miguel, Q. Shen, Hard, flexible and dynamic constraint satisfaction, Knowledge Engineering Rev. 14 (3) (1999) 199–220.
[21] I. Miguel, Q. Shen, Dynamic flexible constraint satisfacton, Appl. Intelligence 13 (3) (2000) 231–245.
[22] I. Miguel, Q. Shen, Solution techniques for constraint satisfaction problems: Advanced approaches, Artificial Intelligence Rev. 15 (2001) 269–293.
[23] I. Miguel, Dynamic flexible constraint satisfaction and its application to ai planning, Ph.D. Thesis, Division of Informatics, Edinburgh University, 2001.
[24] I. Miguel, Q. Shen, P. Jarvis, Efficient flexible planning via dynamic flexible constraint satisfaction, Engrg. Appl. Artificial Intelligence 14 (3) (2001) 301–327.
[25] S. Minton, M. Johnston, A. Philps, P. Laird, Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems, Artificial Intelligence 58 (1992) 161–205.
[26] J. Moura Pires, F. Moura Pires, R. Almeida Ribeiro, Structure and properties of leximin fcsp and its influence on optimisation problems, in: Proc. 7th Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, 1998, pp. 188–194.
[27] W. Pedrycz, F. Gomide, An Introduction to Fuzzy Sets: Analysis and Design, MIT Press, Cambridge, MA, 1999.
[28] P. Prosser, An empirical study of phase transitions in binary constraint satisfaction problems, Artificial Intelligence 81 (1996) 81–109.
[29] T. Schiex, Arc consistency for soft constraints, in: Proc. 6th International Conference on Principles and Practice of Constraint Programming, Singapore, 2000, pp. 411–424.
[30] T. Schiex, H. Fargier, G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, in: Proc. IJCAI-95, Montreal, Quebec, 1995, pp. 631–637.
[31] T. Schiex, G. Verfaillie, Nogood recording for static and dynamic constraint satisfaction problems, Internat. J. Artificial Intelligence Tools 3 (2) (1994) 187–207.
[32] S. Smith, D. Nau, T. Throop, Total-order multi-agent task-network planning for contract bridge, in: Proc. AAAI-96, Portland, OR, 1996, pp. 108–113.
[33] E. Tsang, Foundations of Constraint Satisfaction, Academic Press, London, 1993.
[34] P. van Hentenryck, T. Provost, Incremental search in constraint logic programming, New Generation Computing 9 (1991) 257–275.
[35] G. Verfaillie, T. Schiex, Solution reuse in dynamic constraint satisfaction problems, in: Proc. AAAI-94, Seattle, WA, 1994, pp. 307–312.
[36] R. Wallace, Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs, in: Proc. IJCAI-93, Chambéry, France, 1993, pp. 239–245.
[37] R. Wallace, E. Freuder, Stable solutions for dynamic constraint satisfaction problems, in: Proc. 4th International Conference on Principles and Practice of Constraint Programming, Pisa, Italy, 1998, pp. 447–461.
[38] D. Weld, Recent advances in AI planning, AI Magazine 20 (2) (1999) 93–123.
[39] D. Wilkins, K. Myers, A multiagent planning architecture, in: Proc. 4th International Conference on Artificial Intelligence Planning Systems, Pittsburgh, PA, 1998, pp. 154–162.