Heuristics for Fuzzy Constraint Satisfaction

Hans W. Guesgen **Anne Philpott**

Computer Science Report No. 123
July 1995

Heuristics for Fuzzy Constraint Satisfaction

Hans W. Guesgen
Computer Science Department
University of Auckland
Private Bag 92019
Auckland, New Zealand
hans@cs.auckland.ac.nz

Anne Philpott
Programmable State Controller Section
Production Machinery Limited
Fisher & Paykel
Unit 4, Fencible House, 18 Fencible Drive
Howick, Auckland, New Zealand
philpota@fp.co.nz

Abstract

Work in the field of AI over the past twenty years has shown that many problems can be represented as constraint satisfaction problems and efficiently solved by constraint satisfaction algorithms. However, constraint satisfaction in its pure form isn't always suitable for real world problems, as they often tend to be inconsistent, which means the corresponding constraint satisfaction problems don't have solutions.

A way to handle inconsistent constraint satisfaction problems is to make them fuzzy. The idea is to associate fuzzy values with the elements of the constraints, and to combine these fuzzy values in a reasonable way, i.e., a way that directly corresponds to the way how crisp constraint problems are handled.

The purpose of this paper is to briefly introduce a framework for fuzzy constraint satisfaction problems and to discuss some heuristics for solving them efficiently.

1 Introduction

One of the research areas in AI that has gained increasing interest during recent years is constraint satisfaction, mainly because many problems like searching, scheduling, planning, etc. can be formulated as constraint satisfaction problems (CSPs) and can be solved by constraint satisfaction methods in an efficient way. A constraint satisfaction problem can be defined as follows: Given a set of variables $V_1, \ldots, V_m$ over domains $D_1, \ldots, D_m$, respectively, and a set of constraints $C_1, \ldots, C_n$, each constraint ranging over a subset of $V_1, \ldots, V_m$, find an assignment of values $(d_1, \ldots, d_m) \in D_1 \times \cdots \times D_m$ such that $C_1, \ldots, C_n$ are satisfied.

Solving a problem by using constraint satisfaction methods assumes, however, that the underlying problem is solvable at all, which isn't necessarily the case. On the contrary, experience has taught us that many real-world problems are inconsistent and thus do not have a solution. Nevertheless, we as human beings are able to cope with this situation. The way we usually deal with inconsistent problems is the following:

1. Find a problem that is both consistent and closely related to the original problem.
2. Solve the related problem.
3. Take the solution of the related problem as an 'almost' solution for the original problem.

In particular, if it is not possible to satisfy the entire set of constraints, the overconstrained problem can be relaxed by switching off or weakening some constraints in such a way that the relaxed problem is closely related to the original problem and has a solution.

Without additional knowledge, it is almost impossible to automatically transform an overstrained constraint satisfaction problem into a weaker problem in an intuitively plausible manner. In particular, the question is which constraints to relax and how to relax them.

There are several approaches to constraint relaxation, ranging from theoretical formulations as in [11, 18] to practical applications as in [5, 9]. All these approaches have in common that they attack a given CSP by finding a solution of a relaxed CSP that differs only minimally from the original CSP. The difference is expressed in terms of a metric.

There are various ways to define a metric on a given CSPs, i.e., to state how far a relaxed CSP is from the original CSP and with this how far away the approximate solution is from the ideal one. In [15], for example, we have used the concept of penalties. Values not being in the original constraint relations are marked by natural numbers greater than 0. More recently, fuzzy set theory has been used to capture the idea of constraint relaxation [7, 23].

In principle, fuzzy constraint satisfaction provides us with a powerful framework for constraint relaxation. However, fuzzy constraint satisfaction is also computationally expensive, as it converts a classical constraint satisfaction problem into an optimization problem. Heuristic search is an option to cope with the additional complexity.

The field of constraint satisfaction offers a variety of heuristic search methods. This paper investigates how heuristic search methods for classical constraint satisfaction can be applied to fuzzy constraint satisfaction. In particular, the rest of this paper is structured as follows. Section 2 starts with an introduction of the underlying concepts, like fuzzy constraints, fuzzy constraint networks, α-solutions, etc. Section 3 sketches a branch and bound algorithm for fuzzy CSPs. Sections 6 and 7 discuss heuristics for fuzzy constraint satisfaction. Section 8 summarizes the paper.

2 Fuzzy Constraints

To keep this paper self-contained, we briefly review in this section the basic concepts of fuzzy constraint satisfaction as introduced in [14]. These concepts are closely related to the ones in [7], the main idea of which is to ignore constraints in the network, if they can't be satisfied. Some constraints can be ignored more easily than others. To express how easy it is to ignore a constraint, each constraint C is associated with a priority degree α_C ranging in the scale $[0, 1]$. $1 - \alpha_C$ then indicates to what extend it is possible to violate C.

Given a network of constraints with priorities, an assignment of values $(d_1, \ldots, d_m)$ to the variables $V_1, \ldots, V_m$ of the constraint network can be associated with some fuzzy membership grade. This membership grade is computed from the priorities of the constraints. If a constraint C is not satisfied by $(d_1, \ldots, d_m)$, then $1 - \alpha_C$ limits the membership grade of the solution.

Unlike [7], we don't associate a fixed priority with each constraint of the network but define a constraint as being more or less satisfied by some given assignment of values. For example, instead of stating *The color of the object is supposed to be red* with priority 0.75 and then returning a membership grade of 1 if the object is indeed red or $1 - 0.75 = 0.25$ if the object is not red, we proceed as follows: There is a constraint *The color of the object must be red* which has the same priority as the other constraints in the networks. However, this constraint may be more or less satisfied. If the color of the object is red, then the constraint is satisfied with degree 1; if the color of the object is burgundy, then the constraint might be satisfied with degree 0.75; if the color is pink, then with degree 0.5; and so on.

In other words, a fuzzy constraint network $\tilde{N}$ consists of a set of fuzzy variables $\tilde{V}_1, \ldots, \tilde{V}_m$ in the (crisp) domains $D_1, \ldots, D_m$, respectively, and a set of fuzzy constraints $\tilde{C}_1, \ldots, \tilde{C}_n$, each $\tilde{C}_i$ $(i = 1, \ldots, n)$ ranging over a subset of $\tilde{V}_1, \ldots, \tilde{V}_m$. The relation of $\tilde{C}_i$ is a fuzzy relation in the product space of the subset of $D_1, \ldots, D_m$ that corresponds to the fuzzy variables of $\tilde{C}_i$. (See [14] for more details.)

The main operations performed on the relations represented by a constraint network are union and intersection: A constraint may be viewed as the union of one-element sets, each element in such a set representing a possible choice of values for the variables of the constraint, i.e., an assignment that satisfies the constraint. A constraint network, on the other hand, may be viewed as the intersection of the relations represented by the constraints of the network.

There are several ways to define the intersection and union of fuzzy sets. For the reasons discussed in [14], we adopted the original min/max combination scheme [25] for combining fuzzy constraint relations, i.e., for defining the relation that is represented by a fuzzy constraint network. The heuristics introduced in this paper assume that the min/max combination scheme is used.

In most cases, we are not interested in computing the entire relation represented by a fuzzy constraint network, but want to obtain an element of this relation whose membership grade is beyond a certain threshold α. Such an element is called an α-solution of the fuzzy constraint network.

During the recent years, a great diversity of methods has been developed for solving traditional CSPs. They can generally be divided into two broad categories:

1. Consistency propagation algorithms which attempt to achieve various levels of consistency, progressing from local consistency to eventual network consistency (see, for example, [19] and follow-up papers).

2. Constructive heuristic search methods. These are commonly grounded in a basic backtrack (BT) search and enhanced by strategies for making choices about forward or backward moves, avoiding redundant checking, or ordering variables or values in a way that might expedite search (see, for example, [22]).

In the following sections, we will look at some of these methods. In particular, we will address variable and value ordering heuristics, forward checking as an example of a strategy for making forward moves, and arc consistency as an example of a consistency propagation algorithm.

3 Backtracking

We will start our discussion of constructive heuristic search methods with a review of the basic tree search algorithm that underpins many of the heuristically enhanced search processes: backtracking. Backtracking (BT) is an accepted classic algorithm for solving CSP. In BT search variables are instantiated one after the other. Each instantiation is validated by performing consistency checks backwards against the past variables. If no consistent value can be found for a variable, then the previous variable is uninstantiated and a new value sought for that. BT search finds the first consistent instantiation of all the variables, if such an instantiation exists, or can be continued to find all consistent instantiations.

In fuzzy constraint satisfaction consistency is a matter of degree, and it is more than likely that some level of optimality is sought in the consistency of an acceptable solution. This introduces major variations to BT search. Searching must continue after a consistent instantiation is found, if the solution is not 'good enough'. However, some savings can be made in any continued search by pruning search paths that are provably no better than the current best instantiation.

This is in fact the common optimization technique branch and bound (B&B). Freuder [13] adopted B&B as a natural choice in seeking an analogue of backtracking to find optimal solutions for partial CSPs. Ruttkay [23] suggests that in constructing solutions for fuzzy CSP heuristic, BT search can be replaced by B&B search.

B&B search operates in the same way as BT search with the two variations previously mentioned. The best solution so far is recorded, and a search path is abandoned when it is clear that it cannot lead to a better solution. Search stops when all search paths have been either explored or abandoned, or when a perfect solution has been found. In the case of fuzzy constraint satisfaction a perfect solution would be a 1-solution. Search beyond this would be pointless as no better solution can exist. Thus the algorithm defines a depth first search, with chronological BT whenever any search path cannot improve on the best solution found so far.

Empirical tests have shown that pure chronological BT isn't a feasible approach to solving real-world CSPs, and therefore many different heuristics have at various times been combined with BT search to provide better solution methods for CSPs. We will address some of them in the following sections. A more detailed description of heuristic search methods for fuzzy constraint satisfaction can be found elsewhere [21].

4 Variable Ordering Heuristics

It is generally accepted that the order in which variables are instantiated can have a tremendous impact on the size of the search space a backtrack search will explore. The problem of finding a variable ordering that minimizes the search space is very difficult, so most research in this area has been aimed at developing heuristics which reduce the search space.

In this section, we will look first at some of the variable ordering heuristics which have been developed and tested in the area of classical constraint satisfaction, as they are a good starting points in the development of effective variable ordering heuristics for fuzzy constraint solution methods. Then we will consider how they can be transferred from classical to fuzzy constraint satisfaction.

4.1 Variable Ordering Heuristics for Classical CSPs

A considerable amount of research exists on variable ordering heuristics for classical CSPs. Many of these are based on the general idea of instantiating the most difficult or constrained variables first. This is justified by the fact that searching first in the most difficult parts of the search space helps to make the failures appear early in the search. Giving less constrained variable values first can result in costly backtracking when failure due to the constrained variable is discovered late in the search.

Variable ordering methods can be either static or dynamic. Static ordering methods order variables before the search starts. This has the advantage that no overhead during search is required. The disadvantage is, of course, that the orderings do not reflect the changing situation as search progresses. Dynamic variable ordering heuristics overcome this problem by applying the selection methods during the search process, selecting the next variable to instantiate on some basis that takes into account the current state of the search. Dynamic variable ordering methods can add varying degrees of overhead to the search process.

Both static and dynamic variable ordering heuristics have been developed. The following are some of commonly applied variable ordering methods:

Minimum width is a static ordering heuristic based on the connectivity of the network. The heuristic is applied in a preprocessing phase to achieve an ordered list of variables. The list of variables is built from last to first by selecting at each stage a variable which has minimal degree in the subgraph restricted to unselected variables. This selection process results in a minimum width ordering. See [10] for details.

Maximum degree is also a static ordering heuristic which uses the connectivity of the network. Maximum degree aims at a minimum width ordering. Though it may not necessarily achieve such an ordering, it does less work than the minimum width heuristic. Variables are simply ordered in decreasing order according to their degree (or connectivity) in the constraint graph. See [24] for details.

Maximum cardinality is used in a preprocessing phase to achieve a static ordering. Maximum cardinality also exploits connectivity but in a rather different way to minimum width and maximum degree. Variables are selected on the basis of their connections with already selected variables. The first variable is selected randomly. Subsequently, a variable is selected if it is connected to the largest set of already selected variables. See [3] for details.

Depth first search generates a depth first search ordering by a depth first traversal of the constraint graph. The advantage of using such an ordering occurs when search fails and must backtrack. A depth first search ordering ensures that the node backtracked to shares a constraint relation with the node where search has failed. See [3] for details.

Dynamic search rearrangement is, as the name suggests, a dynamic ordering strategy. It is a look ahead scheme which selects as the variable to be instantiated next, the one which has the minimal number of values consistent with the current partial solution. The look-ahead required to make this choice could involve considerable overhead, as it necessitates filtering from the domain of all uninstantiated variables any values not consistent with the current partial solution. See [17] for details.

4.2 Variable Ordering Heuristics for Fuzzy CSPs

The idea behind variable ordering heuristics in a crisp domain transfer directly to a fuzzy domain. To avoid the need to backtrack, the search should attempt to instantiate the most constrained, or most restricting variables first. Some variations on the definition of most restricting, however, may be worth considering for a fuzzy domain.

Grade greater than lower bound is a variation that only counts values with a membership grade greater than some current lower bound. Sharing a constraint relation with a large number of other variables still makes a variable restricting, as does the availability of few values for the variable. However, domain size ordering of variables in a fuzzy environment should not consider a value if its membership grade is below the current lower bound. The same applies if dynamic search rearrangement is transferred to a fuzzy network. The next variable to be instantiated would be the one with the smallest number of values, with consistency greater than the current lower bound, which are consistent with the current partial solution.

Sum of satisfaction ratings is an entirely different alternative [23]. It defines the difficulty of a variable as the sum of the best possible satisfaction ratings for each of the values in the variable domain. Low sums are an indication of availability of few good values for the variable. Values with maximum consistency less than or equal to the current lower bound can be excluded from the summation.

Lowest maximum satisfaction rating looks at the variables from a different viewpoint: The most limiting variables are those which limit the upper bound on the consistency of a solution. These are the variables which participate in the constraint with the lowest maximum satisfaction rating. Instantiating such variables early ensures that search has a realistic measure of the consistency of the solution being built and allows upper bounds to be adjusted right from the start. It is also likely that checking variables with less consistent instantiations early in the search would increase the chances that the lower bound is reached higher in the search tree, and more fruitless search is pruned.

5 Value Ordering Heuristics

Considerably fewer heuristics have been applied to the task of ordering the values available for selection within the domain of a variable. Dechter and Pearl [4] suggest this is partly because if a BT search for all solutions is being performed, the search tree produced is invariant on the value selection. They point out, however, that the situation differs considerably if only one solution is required. In this case the ordering in which values are selected can have a profound effect on the performance of the algorithm. Dechter and Pearl tested the effects of using different levels of information to order the values for selection. They found out that more benefit was obtained from a fairly weak level of look-ahead.

5.1 Value Ordering Heuristics for Fuzzy CSPs

Value ordering in a fuzzy domain may not necessarily have the same aims as value ordering in crisp CSPs. It is true that maximizing future options is still a worthwhile goal but it is important to remember that fuzzy CSPs are optimization problems. Unlike the values in crisp domains, the fuzzy values that are to be ordered already have in their membership grade a metric indicating their suitability. In solving a fuzzy CSP, the sooner a solution with a consistency close to optimality is found, the more the search space is able to be pruned. This means that values are to be ordered according to some measure of their contribution to the optimality of a solution. This can be done either statically or dynamically.

Static maximum consistency fits in very well with consistency filtering [7]. After a fuzzy constraint network has been filtered for local consistency, the membership grade attached to each value in a variable domain is the consistency obtained with that value from the best possible satisfaction of the individual constraints referring to that variable. Ordering values according to this measure therefore requires no extra work if the network has been filtered to a locally consistent state. Values can simply be selected in reducing order of their membership grade. This method provides a static ordering of values before search commences.

Dynamic maximum consistency is a variant of the previous heuristic. In some search situations it may make more sense to order values only when the variable is about to be instantiated. This would have the advantage of tailoring the ordering to the partial solution already instantiated, but is only of benefit if the domains themselves change during the search process. Such a situation occurs if, for example, a forward checking heuristic [22] is being used. Again ordering by decreasing membership grades is sensible as this aims at a best solution first.

It is worth noting, however, that there may be some situations in which ordering values for selection is of no benefit. For example, if there exists an partial α-solution, all values in the domain of any uninstantiated variables whose membership grades are greater than α are equal. The grade of the partial solution cannot be increased beyond α, so value ordering can only sensibly be applied to values which are known to affect the grade of the solution, i.e., those with a maximum possible grade less than α.

The domain membership grade only provides information about the best possible satisfaction level. Nothing is known about the range of membership grades available and how they vary from this maximum. If this situation proves to be common, a heuristic incorporating a little more information about range and variance might be useful. Two obvious possibilities are:

Sum of grades orders values according to the sum of the membership grades obtainable from each constraint in which the variable participates, i.e., sums the grades associated with every tuple which includes the value in any particular constraint and minimizes the sum over all of the constraints by which the variable is restricted.

Average satisfaction rating uses the average of the membership grades that result from checking the constraints (rather than the best possible).

Both these heuristics, however, do have disadvantages. Sum of grades, while giving some indication of the choice of future instantiations available, risks drowning quality with quantity. If, for example, optimality is an important aim, six possibilities each with grades of 0.1 are probably not considered equal to one at 0.6. Both methods also incur extra overheads which would require justification by an improvement in search performance.

In conclusion, while value ordering heuristics may not have been widely applied in either the domain of crisp CSP or combinatorial optimization problems, they certainly warrant investigation in devising solution methods for fuzzy CSP. Early discovery of solutions with are close to optimal membership grades is influenced by the order in which values are selected. Such early optimality is effective in both pruning the search space and in providing relatively good solutions early if search is constrained by time.

6 Forward Checking

Another way of pruning the search space, and with that avoiding unnecessary backtracking, is forward checking [17], which has been shown to be a particularly successful heuristic. Most empirical studies of constructive heuristic search in constraint satisfaction credit this heuristic with being the most effective [12, 22].

FC performs a consistency check each time a variable is instantiated. When a variable V_i is to be instantiated with a value d_i, the algorithm looks at all the uninstantiated variables which share a constraint with V_i and removes from their domains any values inconsistent with d_i. If, in removing these values from the domain of some other variable V_j, the domain becomes empty, this signals the futility of extending the solution with this instantiation. Any changes made to domains by forward checking must then be retracted, and a new instantiation for V_i can be tried.

The strategy of early identification of fruitless search paths by filtering the domains of uninstantiated variables can be applied equally well to fuzzy CSPs. Freuder and Wallace [13] suggest that prospective techniques like forward checking combine well with branch and bound search. They provide a method of discovering the implications of proceeding from the current search point, and thus increase the pruning potential of the branch and bound strategy. The difference in a fuzzy environment arises from the fact that the domain we are filtering is a fuzzy rather than a crisp set. In filtering a crisp set of values, a value is either consistent and can remain in the set, or is inconsistent and can be deleted. Filtering a fuzzy set, however, can alter the membership grade associated with a value by reducing it. If the membership grade reduces to less than or equal to the current lower bound there is no point retaining that value in the set, as it cannot participate in an improved solution.

Figure 1 sketches a forward checking algorithm for fuzzy CSPs.[1] Whenever BT has instantiated a fuzzy variable $\tilde{V}_i$ with a fuzzy value $\langle d_i, \mu(d_i)\rangle$, forward_check is applied to any uninstantiated variable $\tilde{V}_j$. forward_check$(\tilde{V}_i, \tilde{V}_j, d_i)$ performs the filtering of the fuzzy domain of $\tilde{V}_j$ based on the instantiation $\tilde{V}_i \leftarrow \langle d_i, \mu(d_i)\rangle$. If there exists a constraint between $\tilde{V}_i$ and $\tilde{V}_j$ then the membership grades of all values in $\tilde{D}_j$ are altered to reflect the instantiation $\tilde{V}_i \leftarrow \langle d_i, \mu(d_i)\rangle$. Any values whose membership grades become less than or equal to some given lower bound are deleted from the domain. To permit backtracking, FC must also maintain a record, *restricted*$_i$,

[1]For simplicity, we restrict ourselves here to binary constraint networks. This restriction isn't necessary in general.

```
forward_check($\tilde{V}_i, \tilde{V}_j, d_i$)
    if constraint_exists($\tilde{V}_i, \tilde{V}_j$)
        $new\_\tilde{D}_j \leftarrow \emptyset$
        $domain\_changed \leftarrow$ false
        for each $d_j \in D_j$ do
            $new\_grade \leftarrow$ constraint_check($\tilde{V}_i, \tilde{V}_j, d_i, d_j$)
            if $new\_grade < \mu(d_j)$
                $domain\_changed \leftarrow$ true
            if $new\_grade > lower\_bound$
                $new\_\tilde{D}_j \leftarrow new\_\tilde{D}_j \cup \{\langle d_j, new\_grade \rangle\}$
        if $domain\_changed$
            $restricted_i \leftarrow restricted_i \cup \{\tilde{V}_j\}$
            $\tilde{D}_j \leftarrow new\_\tilde{D}_j$
        return $\tilde{D}_j \neq \emptyset$
    else return true
```

Figure 1: Forward checking in fuzzy CSPs.

indicating which domains forward checking $\tilde{V}_i$ has altered. If the domain of $\tilde{V}_j$ is altered in any way by forward_check($\tilde{V}_i, \tilde{V}_j, d_i$) then $\tilde{V}_j$ is added to *restricted*$_i$.

7 Arc Consistency

Forward checking can be taken a step further by applying it iteratively. As a result, we obtain a constraint network which is arc consistent, i.e., in which the domains of each pair of variables are consistent with the constraint between the variables.

Mackworth [19] proposed several algorithms, one of which is AC3, to transform constraint networks into arc consistent constraint networks. Since then, a considerable amount of research has been devoted to improving Mackworth's algorithms. Mohr and Henderson [20] developed AC4 which improves on the worst case performance of AC3, though in fact often does worse in the average case. AC5 [6] achieves improvements for special classes of constraints, and AC6 [2] combines the optimal worst case behavior of AC4 and an average case behavior improved from AC3. Most recently Freuder [12] has reduced constraint checks by using meta level knowledge to infer support. He formulates a general arc consistency algorithm, AC7, which does not depend on the special properties of a limited class of constraints.

Dubois et al. [8] adapted the AC3 algorithm to a fuzzy domain. Their algorithm, called FAC3, suffers from the same redundancies as AC3, and should be amenable to the same sorts of improvements apparent in AC4, AC6, and AC7. There is, however, one major difference which makes support-based algorithms like AC6 and AC7 less appropriate for transfer to a fuzzy environment. Both these algorithms stop checking as soon as they find support for a value, and only look for more support if the original support is deleted. This methodology does not transfer to a fuzzy environment because, to achieve a consistency grade which reflects the maximum achievable for any value in a fuzzy domain, every consistent pair must be checked.

Figure 2 displays a new fuzzy arc consistency algorithm which is proposed as an alternative to FAC3. The aim in developing the algorithm is to incorporate some of the savings in constraint checks achieved in AC4–7, but bearing in mind the need to check every pair with consistency

```
new_FAC($\tilde{N}$)
    $Q \leftarrow$ set of all constraints in $\tilde{N}$
    while $Q \neq \emptyset$
        $\tilde{C}_{ij} \leftarrow \mathsf{next}(Q)$
        $new\_\tilde{D}_i \leftarrow \emptyset$
        $new\_\tilde{D}_j \leftarrow \emptyset$
        for each tuple $(x, y) \in C_{ij}$
            $\mu(x, y) \leftarrow \min\{\mu(x), \mu(y), \mu(x, y)\}$
            if $\mu(x, y) < lower\_bound$
                delete $(x, y)$ from $\tilde{C}_{ij}$
            else
                $new\_\tilde{D}_i \leftarrow new\_\tilde{D}_i \cup \{\langle x, \mu(x)\rangle\}$
                $new\_\tilde{D}_j \leftarrow new\_\tilde{D}_j \cup \{\langle y, \mu(y)\rangle\}$
        if $new\_\tilde{D}_i \neq \tilde{D}_i$
            $Q \leftarrow Q \cup$ all constraints involving $i$
            $\tilde{D}_i \leftarrow new\_\tilde{D}_i$
        if $new\_\tilde{D}_j \neq \tilde{D}_j$
            $Q \leftarrow Q \cup$ all constraints involving $j$
            $\tilde{D}_j \leftarrow new\_\tilde{D}_j$
```

Figure 2: New fuzzy arc consistency algorithm.

greater than the lower bound, and without adding large and complex data structures to keep track of information.

The algorithm is successful by using first upward then downward propagation of consistency values. The information that needs to be recalled is stored by updating the explicit representation of the constraint, rather than introducing separate structures to store it. Tuples that have their consistency grade reduced to less than the lower bound are deleted and do not need to be reconsidered if the constraint is reconsidered because of changes elsewhere. The algorithm builds new domains rather than updating the existing ones for two reasons:

1. This allows each checked value to be incorporated as a fuzzy union operation in a straightforward way and leaves the initial unary relations unchanged and available for intersection with subsequent tuples involving the same values.

2. The fuzzy union operation itself requires less work if it starts with an empty set.

The algorithm provides some definite advantages over the fuzzy version of the AC3 algorithm. One of the advantages is bidirectionality. As with AC7, this algorithm makes the property of undirectedness explicit. Both domains are updated after each constraint check. The algorithm therefore does not perform the redundant constraint check that FAC3 does to update the second domain. This reduces the constraint checks performed by at least half.

Another advantage lies in how the consistency checks are performed. As each constraint is a subset of the Cartesian product of the variable domains, the algorithm can save other constraint checks also. By looking up each tuple of the constraint, it does at worst the same amount of work as the other algorithms which look up every combination of the domain values. If the constraints are more restrictive than the domains, it may however do considerably less work. Domain values unsupported by the constraint relation are never checked at all.

8 Conclusion

In the first part of this paper, we briefly introduced a framework for fuzzy constraint satisfaction, which provides a general basis for fuzzy constraint satisfaction algorithms. This framework has also been applied in the area of spatial and temporal reasoning. In particular, we applied it to Allen's temporal logic [1] for reasoning about fuzzy spatial and temporal relations. This work is described elsewhere [16].

In the second part of the paper, we discussed some heuristics for solving fuzzy CSPs. We selected those heuristics that are among the most promising ones. In particular, we presented variable and value ordering heuristics, forward checking, and arc consistency in the context of fuzzy constraint networks.

We implemented all heuristics in a Lisp environment and tested them with several fuzzy constraint satisfaction problems [21]. As with the tests in [22], our tests are not exhaustive and allow for further research.

Acknowledgements

Part of this work was supported by the University of Auckland Research Fund under the grant number A18/XXXXX/62090/F3414025. Many thanks to Peter Gibbons for his comments on an earlier version of this paper.

References

[1] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26:832–843, 1983.

[2] M. Cordier and C. Bessiere. Arc consistency and arc consistency again. In *Proc. AAAI-93*, pages 108–113, Washington, DC, 1993.

[3] R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.

[4] R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1987.

[5] Y. Descotte and J.C. Latombe. Making compromises among antagonist constraints in a planner. *Artificial Intelligence*, 27:183–217, 1985.

[6] Y. Deville and P. van Hentenryck. An efficient arc consistency algorithm for a class of csp problems. In *Proc. IJCAI-91*, pages 325–330, Sidney, Australia, 1991.

[7] D. Dubois, H. Fargier, and H. Prade. Propagation and satisfaction of flexible constraints. Rapport IRIT/92-59-R, IRIT, Toulouse Cedex, France, 1992.

[8] D. Dubois, H. Fargier, and H. Prade. Propagation et satisfaction de constraintes flexibles. In R.R. Yager and L. Zadeh, editors, *Fuzzy Sets, Neural Networks and Soft Computing.* Kluwer, Dordrecht, The Netherlands, 1993.

[9] B.N. Freeman-Benson, J. Maloney, and A. Borning. An incremental constraint solver. *Communications of the ACM*, 33:54–63, 1990.

[10] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29:24–32, 1982.

[11] E.C. Freuder. Partial constraint satisfaction. In *Proc. IJCAI-89*, pages 278–283, Detroit, Michigan, 1989.

[12] E.C. Freuder. Using metalevel constraint knowledge to reduce constraint checking. In *Proc. ECAI-94 Workshop on Constraint Processing*, pages 27–33, Amsterdam, The Netherlands, 1994.

[13] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[14] H.W. Guesgen. A formal framework for weak constraint satisfaction based on fuzzy sets. In *Proc. ANZIIS-94*, pages 199–203, Brisbane, Australia, 1994.

[15] H.W. Guesgen and J. Hertzberg. A constraint-based approach to spatiotemporal reasoning. *Applied Intelligence (Special Issue on Applications of Temporal Models)*, 3:71–90, 1993.

[16] H.W. Guesgen, J. Hertzberg, and A. Philpott. Towards implementing fuzzy Allen relations. In *Proc. ECAI-94 Workshop on Spatial and Temporal Reasoning*, pages 49–55, Amsterdam, The Netherlands, 1994.

[17] R.M. Haralick and G.L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[18] J. Hertzberg, H.W. Guesgen, A. Voß, M. Fidelak, and H. Voß. Relaxing constraint networks to resolve inconsistencies. In *Proc. GWAI-88*, pages 61–65, Eringerfeld, Germany, 1988.

[19] A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[20] R. Mohr and T.C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[21] A. Philpott. Fuzzy constraint satisfaction. Master's thesis, University of Auckland, Auckland, New Zealand, 1995.

[22] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9:268–299, 1993.

[23] Z. Ruttkay. Fuzzy constraint satisfaction. In *Proc. FUZZ-IEEE'94*, Orlando, Florida, 1994.

[24] H.S. Stone and J.M. Stone. Efficient search techniques: An empirical study of the n-queens problem. Technical Report RC 12057 (#54343), IBM T.J. Watson Research Center, Yorktown Heights, New York, 1986.

[25] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.