Fuzzy Constraint Satisfaction Using CCM — A Local Information Based Computation Model

Yasusi Kanada
Tsukuba Research Center, Real World Computing Partnership
Takezono 1-6-1, Tsukuba, Ibaraki 305, Japan
E-mail: *kanada@trc.rwcp.or.jp*, WWW: *http://www.rwcp.or.jp/people/yk/*

Abstract

The present paper proposes a method of solving the fuzzy constraint satisfaction problems defined by Ruttkay. This method is based on CCM, which is a computational model for emergent computation, or for locality-based problem solving. CCM is a type of production system. It works stochastically, or randomly, and works with evaluation functions that are computed only with local information. CCM has already been applied to constraint satisfaction problems (CSPs). Binary-valued evaluation functions, each of which indicates whether a constraint is satisfied, are used. If the values of the evaluation functions are extended to real values, fuzzy CSPs can be expressed in CCM, and solved using a technique similar to GSAT or annealing. We applied this method to a fuzzy graph coloring problem, and evaluated the performance. This method can also be applied to open and dynamic fuzzy/non-fuzzy CSPs, in which data and constraints are changing dynamically or coming from or going to outside the system.

1. Introduction

Conventional constraint satisfaction problems and the methods for solving them are “hard.” If there are even only a few contradictory constraints, the problems becomes unsolvable. However, constraints in the real world usually contain contradictions. Some combinations of specific values are more preferable than others, and constraints may be added or removed dynamically while the problem is being solved. Thus, methods for specifying and solving “soft” constraint satisfaction problems are necessary. Several types of soft constraint satisfaction were proposed by Freuder and Wallace [Fre 92], Ruttkay [Rut 94]. These and several other papers were summarized by Ruttkay. Ruttkay gives several possible definitions of fuzzy constraint satisfaction problems (CSPs), and also offers a method of solving fuzzy CSPs.

In Ruttkay’s definition, the degree of satisfaction of each constraint, c_i, satisfies $0 \le c_i \le 1$. If the constraint is fully satisfied, then $c_i = 1$, and if it is fully violated, then $c_i = 0$. The degree of joint satisfaction of constraints c_1, c_2, …, c_N is defined in one of the following three methods.

1. Conjunctive combination:
$$C_{min} = min\left\{ c_i \mid i = 1, \ldots, N \right\}$$
2. Productive combination: $$C_{pro} = \left(\prod_{i=1}^{N} c_i \right)^{\frac{1}{N}}$$
3. Average combination: $$C_{ave} = \frac{1}{N} \sum_{i=1}^{N} c_i$$

The fuzzy CSPs are the problems of maximizing the degrees of joint satisfaction shown above. The definition of productive combination is slightly modified to achieve normalization. C_{ave} is the arithmetic mean of c_i, and C_{pro} is the geometric mean of c_i. The harmonic mean version of the combination may be added to the list.

Although fuzzy CSPs are types of constrained optimization problems, their nature is different from typical global optimization problems, where the evaluation functions cannot be evaluated only using local information. In fuzzy CSPs, the degree of satisfaction of each constraint is evaluated locally or using only a small number of data.

Ruttkay’s algorithm is based on a branch-and-bound method. Thus, it is suitable for finding the best solution, i.e., the state in which the joint satisfaction takes its maximum value. However, if the problem is large, this algorithm will take an excessive amount of time because the time complexity of the branch-and-bound algorithm is exponential in nature. The time complexity can probably be reduced if the best solution is not required and appropriate branch cuts are performed. However, other types of methods, such as randomized methods, may perform better in finding approximate solutions for fuzzy CSPs. In the case of non-fuzzy CSPs, such non-backtracking methods are successfully used in several applications [Mor 93, Sel 92, Sel 93, Min 92].

A method of solving fuzzy CSPs, based on CCM (the chemical casting model), which is a production-system-based computational model for emergent computation [For 91] or for locality-based problem solving, is shown in the present paper. Local degrees of satisfaction are used as evaluation functions, and partial summations of them are stochastically optimized in this method. A computation model called CCM is briefly explained in Section 2. The CCM-based method of *non*-fuzzy constraint satisfaction, proposed by Kanada [Kan 94a], is explained in Section 3. The CCM-based method of fuzzy constraint satisfaction is then explained in Section 4. This method is applied to a fuzzy coloring problem. The result is shown in Section 5. Finally, conclusions are given in Section 6.

2. Computational Model CCM

CCM (the chemical casting model) [Kan 92, Kan 94a] is explained briefly in the present section.

Real-world problems are not necessarily expressed by static constraints and evaluation functions. New information may be found while solving problems, and preexisting information may be dynamically changed by environmental change. The development of CCM aims at solving problems by self-organizing or emergent computation [For 91] in such situations. Complete information is not usually available beforehand in such situations. Conventional combinatorial problem solving methods that are based on the assumption of complete information are weak when the environment is continually changing. Thus, CCM has been developed for computation based on local and partial information. CCM is based on a production system. Production systems are often used for developing expert systems or modeling human brains. However, CCM is different from conventional production systems. Firstly, evaluation functions, which are evaluated using only local information, are used. Secondly, stochastic control, or randomized ordering of rule applications, is used. Production rules are also applied only using local information.

The system components in CCM are shown in **Figure 1**. The set of data to which the rules apply is called the *working memory*. A unit of data in the working memory is called an *atom*. An atom has a type and an internal state, and may be connected to other atoms by *links*. Links are similar to chemical bondings, but the difference is that links may have directions. Any discrete data structures such as lists, trees, graphs or networks can be represented using atoms and links.

The state of the working memory is changed locally by *reaction rules*. "Locally" means that the number of atoms referred by a reaction rule is small.[1] The reaction rules are written as forward-chaining production rules, such as rules in expert systems. However, reaction rules are at a lower level, or more primitive, than rules in expert systems. So, the reaction rules are more similar to reaction formulae in chemical reactions, and thus, this model is called the *chemical* casting model. The syntax of reaction rules is as follows:

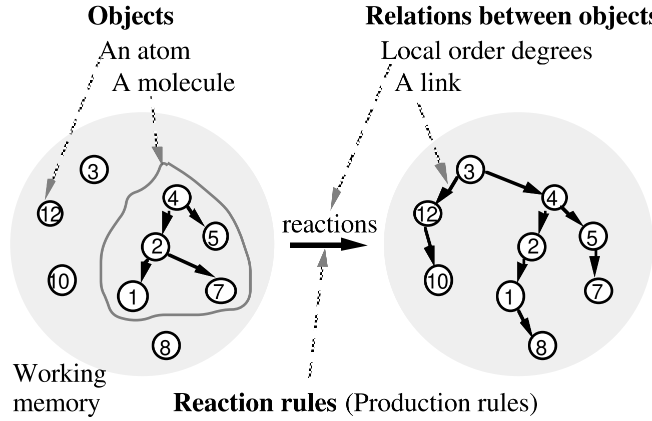


Figure 1. The elements of CCM

LHS → RHS.

The left-hand side (LHS) and the right-hand side (RHS) are sequences of patterns.

For example, the following reaction rule, which is a rough sketch, simulates the generation of water from oxygen and hydrogen:

H-H, O-? → H-O-H, ?

(This approximately means $H_2 + \frac{1}{2}O_2 \rightarrow H_2O$).

There are four patterns both in the LHS and RHS: two H's, an O, and "?" (an unknown atom). Each pattern matches an atom of type oxygen or type hydrogen in the working memory.

The reaction rule can be activated when there is a set of atoms that matches the LHS patterns. If the reaction rule is activated, the matched atoms vanish and new atoms that match the RHS patterns are generated. A single reaction rule is enough for solving a simpler optimization or constraint satisfaction problem like the graph vertex coloring problem, which is described later, or the 0–1 integer programming problem. Two or more reaction rules are needed in more complex systems, in which there are two or more ways of changing atoms.

Local order degrees (LODs) are a type of evaluation functions. LODs express the degrees of local "organization" or "order." They are defined by the user to take a larger value when the local state of the working memory is better. An LOD may be regarded as a negated *energy*. For

[1] Because (physical) distance is not a factor in CCM unlike systems such as a chemical reaction system, "locally" does not mean the distance is small.

example, it is analogous to bonding energy in chemical reaction systems.

A reaction takes place when the following two conditions are satisfied. First, there exists an atom that matches each pattern in the LHS. Second, the sum of the LODs of all the atoms concerned in the reaction, i.e., the atoms that appear on either side of the reaction rule, does not decrease as a result of the reaction. Reactions repeatedly occur while the above two conditions are satisfied by a combination of any rule and atoms. The system stops when such a combination is exhausted. However, reactions may occur again if the working memory is modified because of changes in the problem or the environment. Thus, open and dynamic problems as mentioned beforehand can probably be handled properly using CCM.

Typically, there are two or more combinations that satisfy the two conditions at once. There are two possible causes that generates multiple combinations. One cause is that there are two or more collections of atoms that satisfy the LHS of a reaction rule. The other cause is that there are two or more reaction rules containing atoms that match the patterns in the LHS. In each case, the order of the reactions, or the order of selection of such combinations, and whether they occur in parallel or sequentially is determined stochastically or randomly. Therefore, although the microscopic behavior of CCM, i.e., a reaction, is deterministic, the macroscopic behavior is nondeterministic or random.

3. CCM-based Non-Fuzzy Constraint Satisfaction

An example of non-fuzzy CSP and a method of solving it using CCM are demonstrated in the present section. This example and the method will be extended to a fuzzy CSP and a method of solving it in the next section.

CCM-based problem solving is a biased random walk in the search space [Kan 94a]. Reaction rules are defined as the methods of moving to the neighboring states in the search space. In other words, reaction rules define the neighbors of each state in the search space. LODs, which are defined to indicate which are the locally better states, bias the search. The mean value of the LODs is called the mean order degree (MOD). Although CCM-based systems do not compute MODs, they operate so as to increase MODs or sums of LODs stochastically [Kan 94b]. MODs can be defined on a small, medium or large scale. MODs increase with every scale, if the LODs and rules are defined properly.

Kanada [Kan 94a] describes a method of problem solving using CCM, and uses the *N* queens problem, which is a CSP, for example. In this case, each constraint is expressed as an LOD between two atoms. The value of an LOD is higher when the constraint is satisfied, and it is lower when the constraint is violated. In this way, the global MOD, or the total of LODs, is optimized. That means that all the constraints are satisfied by the system operation. The same method can be applied to other CSPs. A CCM-based system to solve a coloring problem is shown below.

The problem is as follows. The graph vertex coloring problem is a problem of how to color the vertices of a graph using a specified number of colors, for example, four. Each pair of neighboring vertices must be given different colors. A map coloring problem can be converted to a graph coloring problem, if areas of the map are converted to vertices and the area borders are converted to edges. Thus, the map coloring problem can be solved by the same caster as the graph coloring one. For example, the problem of coloring the graph with five vertices, shown in **Figure 2**, is equivalent to the problem of coloring the map with five areas, which is also drawn in Figure 2. The data structure used for solving the problem is as follows. Vertices are represented by atoms. An atom of type vertex has a color as its internal state. c1, c2, c3 and c4 are the colors in Figure 2.

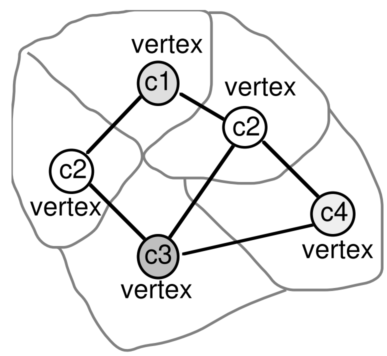


Figure 2. An example of a graph coloring problem and its representation

The reaction rule and LOD to solve the problem are shown below. Firstly, the only reaction rule is shown in a visual form in **Figure 3**.[1] This rule refers only to two neighboring vertices and the edge between them, and it changes the color of one of the vertices randomly. The LHS of the rule contain two patterns. They match atoms of type vertex. There is a link between the atoms, and this link represents the edge between the vertices. Thus, vertex1 and vertex2 can only match two vertices, which have a link between them. When a reaction occurs, the internal state of the atom that matched vertex1, is rewritten. That is, the color, which was C1 before the reaction, becomes C3, which is generated randomly. C3 is selected from predefined colors. Because no constraint between

[1] Reaction rules must be coded by computation language SOOC-94 (Self-Organization-Oriented Computing) to be executed.

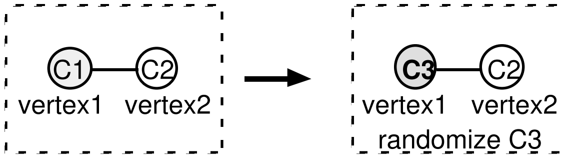


Figure 3. The reaction rule for the graph coloring system

C1, C2 and C3 is given, these colors can be either the same or different.[1,2]

Secondly, the definition of LOD is shown below. The LOD is defined between two vertices, *v1* and *v2*. Its value is 1 when the constraint between *v1* and *v2* is satisfied, and it is 0 otherwise.

$$O(v1, v2) = \begin{cases} 1 & \textbf{if not } connected(v1, v2) \textbf{ or } v1.color \neq v2.color \\ 0 & \textbf{otherwise} \end{cases}$$

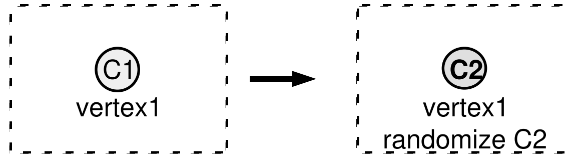
This definition means that the value of the LOD is 1 (higher) if the vertices are not connected or have different colors, and that it is equal to 0 (lower) otherwise. The value of LOD is 1 when the vertices are not connected (because there is no constraint, no constraint is violated), in this case. The value is 1 when the vertices are connected and have different colors, because the constraint between the vertices is satisfied. The value is 0 when the vertices are connected and have the same color, because the constraint is violated. Thus, $O(v1, v2)$ can be regarded as the membership function of the crisp set of pairs of vertices that satisfy the constraints.

In general, a problem to be solved can be expressed in CCM, if a method of walking in the search space can be expressed as a reaction rule, and the problem consists of local constraints, which can be expressed by binary LODs.

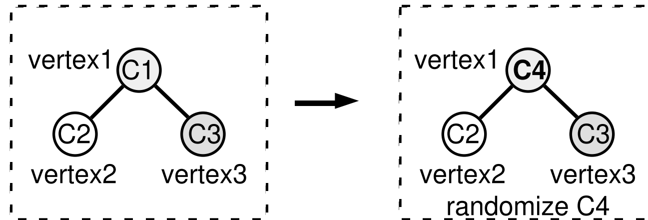
Reactions occur successively so that the values of MODs increase stochastically when the rule shown above is used. If the system reaches a solution state, the system stops, because no reaction can occur in this state. However, a reaction may decrease the LODs of the neighboring edges. So the MOD does not linearly increase, and the system does not necessarily find a solution in finite time. However, experiments show that the execution time is always finite in practice.

A pattern of atoms, whose value is not changed by the rule, is called a *catalyst* [Kan 94a]. In the rule in Figure 3, vertex2 is a catalyst. A rule with no catalyst, as illustrated in **Figure 4** (a), can be given. A rule with two or more catalysts can also be given. A rule with two catalysts is shown in Figure 4 (b). Addition or removal of catalysts changes the locality of the computation, and changes the performance. In the coloring problem, a system with only the no-catalyst rule performs a complete random walk in the search space. The performance is out of the question because this system does not stop even if a solution is found. If more catalysts are added, the number of reactions decreases, and the performance improves under certain conditions. The rules in Figures 4 (a) and (b) have a fixed number of catalysts. However, the number of catalysts can vary dynamically. In Figure 4 (c), all the neighbors of vertex1 are the catalysts, and the number of catalysts depends on the vertex matched to vertex1.

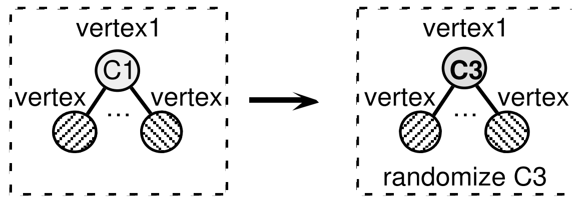
A slightly modified version of the rule and LOD has been coded in SOOC-94, which is a computation language or a general problem solver for CCM-based computation, and then executed. It has been applied to the map of the mainland of the USA, consisting of 48 states [Tak 92], shown in **Figure 5**. All the states are in the same color in the initial state. The result of performance measurement of this system using a SOOC-94 compiler and interpreter on top of Common Lisp on a Macintosh Quadra 840 AV is shown below. A correct solution was found in every run. The number of reactions, the number of LHS matches (including failed cases), and the execution time are measured. The relation between the number of catalysts and the average performance is shown in **Figure 6**. The num-



(a) With no catalyst



(b) With two catalysts



(c) With a variable number of catalysts

Figure 4. The reaction rules with zero or two catalysts

[1] Because C3 is generated randomly, the same color may be selected for C3 as for C1 or C2. However, the IOD does not increase in the case of such a reaction, so the reaction does not occur (See the definition of LOD given later).

[2] The rule is almost symmetrical and reversible, but the randomize statement in the RHS breaks the symmetry of the rule.

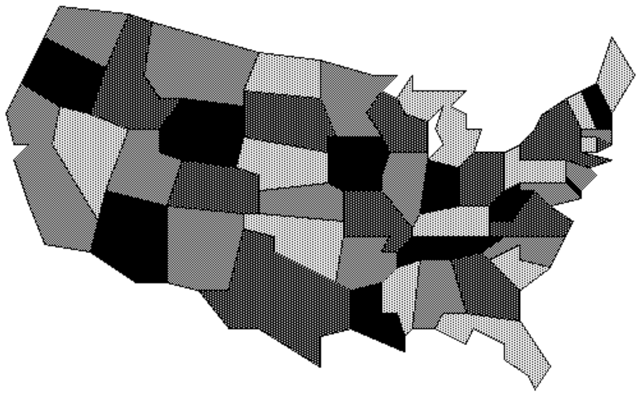


Figure 5. The USA mainland map

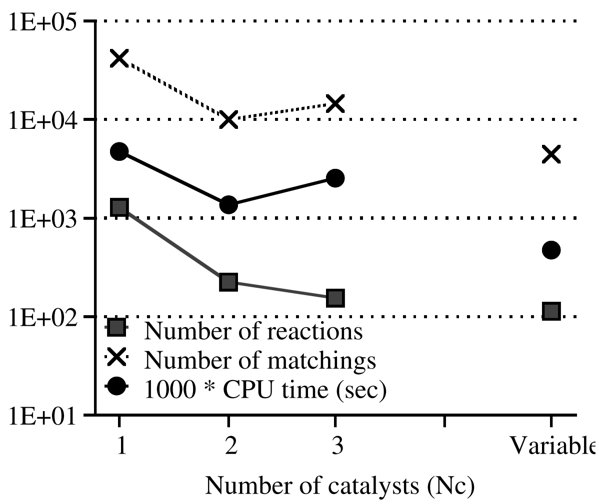


Figure 6. The number of catalysts and the average performance

ber of reactions decreases, but the execution time and the number of LHS matches does not necessarily decrease, when the number of catalysts increases. The average number of catalysts in the variable case is equal to the average number of neighbors, i.e., 4.42, in the USA mainland map.

A particular method for escaping the local maxima of MOD is applied in the variable-catalyst case. This method is called *the frustration accumulation method* (*FAM*) [Kan 94b]. It is similar to a technique used in GSAT [Sel 93]. The variable-catalyst rule sometimes fails to find a solution if this method is not used, but it can almost always find a solution when using this method without significant overhead. The effects of adding or removing catalysts have been analyzed in detail by Kanada [Kan 94b].

4. CCM-based Fuzzy Constraint Satisfaction

The fuzzy graph vertex coloring problem is defined and CCM-based system to solve this problem is shown in the present section.

Ruttkay [Rut 94] describes the fuzzy robot-dressing problem as an example of fuzzy CSP. This problem is very small. Thus, it is suitable for tracking methods of solving fuzzy CSP. However, it is not suitable for evaluating the performance of such methods.

The fuzzy graph vertex coloring problem is the problem of coloring the vertices of a graph using a specified number of colors. A pair of neighboring vertices must be given different colors, and some pairs are preferable to others. The preference is given as a fuzzy constraint between colors. Thus, each degree of preference between colors $c1$ and $c2$, $c(c1, c2)$, is expressed as a real number between 0 and 1. An example of this fuzzy constraint is shown in **Table 1** as a matrix. The degree of preference between the same color is 0, and that between different colors is greater than 0. The preference is symmetric. This means that the preference between colors x and y is the same as that between colors y and x. A fuzzy graph vertex coloring problem can be converted to a fuzzy map coloring problem in the same as in the non-fuzzy coloring problem.

Table 1. An example of a fuzzy color constraint: *c*

Color	Yellow	Blue	Red	Green
Yellow	0	0.8	0.5	0.6
Blue	0.8	0	0.8	0.7
Red	0.5	0.8	0	0.9
Green	0.6	0.7	0.9	0

The same reaction rule as for the non-fuzzy problem can be used for solving the fuzzy coloring problem. If the degree of joint satisfaction is defined as the average combination, C_{ave}, the LOD can be modified as follows.

$$O(v1, v2) = \begin{cases} 1 & \textbf{if not } connected(v1, v2) \\ c(v1.color, v2.color) & \textbf{otherwise} \end{cases}$$

In this case, the mean value of all the LODs is equal to C_{ave}. Thus, it is maximized. $O(v1, v2)$ can be regarded as the membership function of the fuzzy set of pairs of vertices which satisfy the constraints.

If the degree of joint satisfaction is defined as the productive combination, C_{pro}, the LOD can be modified as follows.

$$O(v1, v2) = \begin{cases} 0 & \textbf{if not } connected(v1, v2) \\ \log c(v1.color, v2.color) & \textbf{otherwise} \end{cases}$$

If $c(v1, v2) = 0$, then $\log c(v1, v2) = -\infty$, so a small enough number, e.g., 10^{-99}, must be used for the value of $c(v1, v2)$, instead of 0. In this case, the mean value of all the LODs is approximately equal to $\log C_{pro}$. Thus, C_{pro} is maximized.

However, if the degree of joint satisfaction is defined as the conjunctive combination, this method cannot be applied because C_{min} cannot be expressed by LODs.

If the variable-catalyst rule shown in Figure 4 (c) is used, a technique for escaping from local maxima of MODs is necessary. A modified version of the frustration accumulation method (FAM), which is called *the fuzzy FAM*, is used here. The fuzzy FAM is explained below. Each atom, or vertex, has a type of energy called *frustration*. If a vertex has positive frustration, reactions occur more easily. Each vertex initially has a certain level of frustration, e.g., 0.1. The frustration is increased when the LHS of the reaction rule is tested but there are constraints, concerning to the vertex, which are not or will not be fully satisfied.

More specifically, if the rule and a set of atoms are tested, the reaction does not occur, and the degree of satisfaction of the constraints, c $(0 \le c \le 1)$ is less than 1, the frustration of the atom to be modified by the rule, f, is replaced by $(1 + (1 - c)\,k)\,f$,[1] where k is a constant around 0.005. If the rule and a set of atoms are tested and the reaction occurs but the constraints are not fully satisfied, the frustration is also replaced by $(1 + (1 - c)\,k)\,f$. However, if the reaction occurs and the constraints are fully satisfied, the frustration is reset to its initial value. The reaction occurs when $o_b - f \le o_a$, where o_b is the sum of the LODs of the vertices matched to `vertex1` and other patterns in the rule before the reaction, and o_a is that after the reaction. Thus, the reaction occurs more easily when the value of f is larger.

5. Experiments in Fuzzy Constraint Satisfaction

The performance of the fuzzy version of the system described in the previous section was measured using the USA mainland map. The fuzzy constraint shown in Table 1 was used. All the states were in the same color in the initial state. The system did not stop in reasonable time when rules with a fixed number of catalysts, i.e. 1 to 3, were used. Thus, we can probably conclude that the fuzzy coloring problem is significantly more difficult to solve than the corresponding non-fuzzy problem. When the variable-catalyst rule with the fuzzy FAM is used, fairly good solutions can be found.

[1] $(1 + k)\,f$ is used instead, in the original (non-fuzzy) FAM. If $(1 + k)\,f$ is used in the fuzzy FAM, experiments show that the performance is worse. This probably means that the frustration must increase monotonically when the degree of constraint satisfaction decreases. However, whether the expression $(1 + (1 - c)\,k)\,f$ is the best is not known.

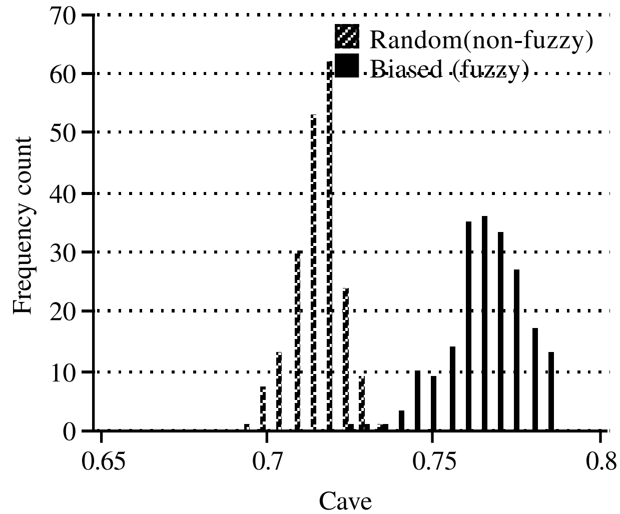


Figure 7. The histogram of C_{ave} in 200 runs

The system with an average combination was tested 200 times with $k = 0.005$. The average, maximum and minimum values of C_{ave} are 0.763, 0.783 and 0.721. The distribution of C_{ave} when the fuzzy version of the system stopped is shown by the continuous line in **Figure 7**. Each class width of C_{ave} is 0.005. All the strong constraints were satisfied in 20 runs, but they were not satisfied in other 180 runs because violation of a strong constraint is not inhibited in the case of an average combination. The broken line is the distribution of C_{ave} when the non-fuzzy version of the system (with $k = 0.4$), described in Section 3, was executed and stopped. This distribution is very sharp and far from normal distributions. There is almost no intersection between these two distributions. This means that the system in the previous section finds much better solutions than the non-fuzzy system that randomly selects a state among the states that satisfy all the strong constraints.

The system with the productive combination was also tested 200 times with $k = 0.005$. The average, maximum and minimum values of C_{pro} are 0.724, 0.764 and 0.669. When the value of $c(v1, v2)$ is 0, it is replaced by 0.03 in this experiment, because if a too small number such as 10^{-99} is used, even the strong constraints cannot be satisfied in most runs.[2] The distribution of C_{pro} when the fuzzy version of the system stopped is shown by the continuous line in **Figure 8**. All the strong constraints were satisfied in 123 runs. They are satisfied in more than half the runs because the penalty for violating a strong constraint is still very high. The (real) values of C_{pro} in the other 67 runs as 0, and these runs are ignored in this

[2] The value for inhibitory color assighnment, 0.03, is near the optimal in our experiments.

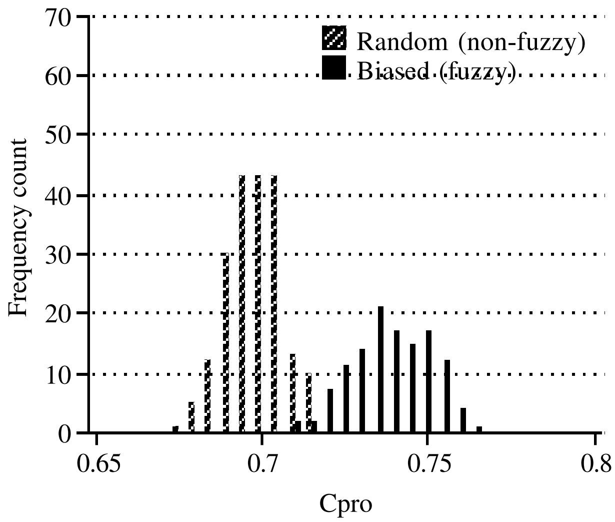


Figure 8. The histogram of C_{pro} in 200 runs

figure. The broken line is the distribution of C_{pro} when the non-fuzzy version of the system (with k = 0.4) was executed and stopped. The performance is similar to that achieved using the average combination.

The execution statistics are compared in **Table 2** with the non-fuzzy CSP using the variable-catalyst rule of Section 3. The average execution time of the fuzzy coloring is much longer than the non-fuzzy coloring. This fact also demonstrates that the fuzzy coloring is much more difficult.

Table 2. The statistics of fuzzy / non-fuzzy USA map coloring using the variable-catalyst rule

	Average number of reactions	Average number of matching	Average exec. time (sec)	Min exec. time (sec)	Max exec. time (sec)
Non-fuzzy coloring	112	4406	0.5	0.3	1.3
Fuzzy coloring (average)	1348	54071	31.4	0.4	272.3
Fuzzy coloring (productive)	659	44630	25.2	1.1	145.7

The above results can be viewed in a different way in **Figure 9**. The system is run repeatedly and the total execution time is measured (using the same data as Figures 7 and 8). The relation between the total execution time and C_{ave} or C_{pro} are plotted in this figure. Two sequences of trials using average combinations are plotted by I and E, and two sequences using productive combinations are plotted by X and Γ. For example, in the average combination case, the first trial took 31.4 seconds, and the resulting C_{ave} was 0.763. When 108.8 seconds were spent

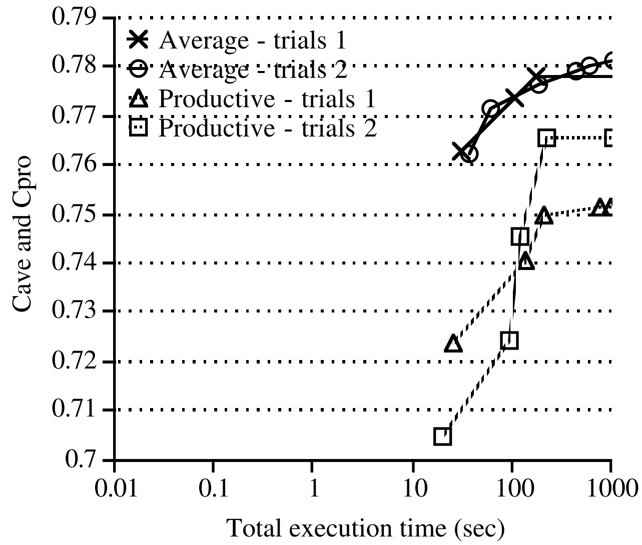


Figure 9. The performance of fuzzy USA map coloring using CCM

in total (in the fourth trial), a better solution, C_{ave} = 0.774, was found, and so on.

Randomly generated constraints are also tested. In these cases, the averages of C_{ave} and C_{pro} are distributed between 0.3 and 0.9, but no other significant difference from the case with the constraints in Table 1 was found.

The CCM-based method is also compared with a backtrack search. All the states are not colored in the initial state in this algorithm. The color that maximizes the sum of the degree of satisfaction of already colored vertices is selected first when coloring a vertex. This algorithm is similar to a best-first search, but it is a *tree-based* depth-first search. Thus, it is different from A* search [Har 68], which is a *queue-based* best-first search.[1] The order of vertices to be colored is fixed. The vertices are sorted before coloring so that more constrained vertices are colored in earlier steps. No dynamic information is used to decide the order of coloring. Thus, the algorithm proposed by Ruttkay, which uses dynamic information, may perform better. Two versions of this algorithm, which use slightly different methods of sorting vertices, were tested. The performance was measured only once for each parameter because these algorithms are deterministic.

The results are shown in **Figure 10**. The result of the first version of the algorithm using the average combination is shown by I. This algorithm found a solution, in which C_{ave} = 0.765, in 0.02 second, and found a better solution in 0.3 second, and so on. The result of the second version using average combination is shown by E. The

[1] A* search has also been tested for solving the fuzzy coloring problem. However, no good evaluation function has been found, and the performance is worse than the tree-based algorithm.

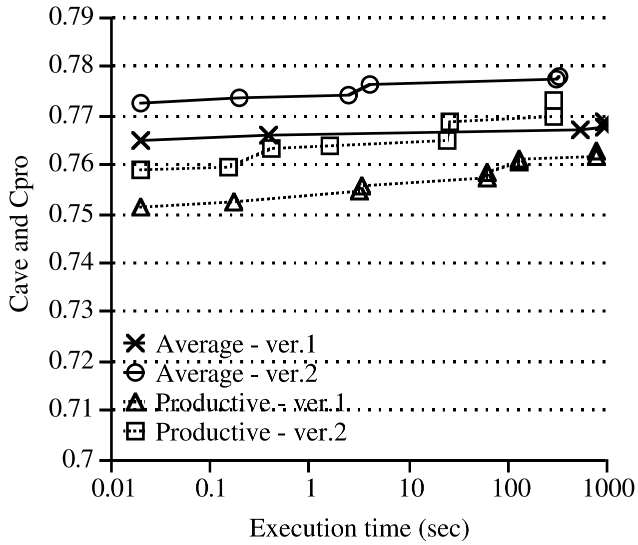


Figure 10. The performance of fuzzy USA map coloring using backtrack search

results of the two versions of the algorithm using productive combination are also shown in Figure 10. Not every run terminated within 1000 seconds.

Figures 9 and 10 can be compared. In the average combination case, the quality of the solutions found using CCM is comparable to that shown in this figure, although it takes more time to get the first solution. However, in the productive combination case, the quality of the solutions found using CCM is worse. This is probably because branch cuts work well in the backtrack search. However, if strong constraints are handled better, CCM-based search may perform better.

6. Conclusion

A method of solving fuzzy constraint satisfaction problems is outlined in the present paper. This method is applied to a fuzzy coloring problem for graphs or maps, and reasonably good solutions can be found in reasonable time, e.g., 20 seconds, in the case of the fuzzy USA map. Although the average execution time of this method is longer than for a backtrack search, it may be improved by parallelization. Our method is much more suitable for parallelization than backtrack algorithms.

This method can also be applied to open and dynamical fuzzy/non-fuzzy CSPs, in which data and constraints are changing dynamically or coming from or going to outside the system.

The main focuses of future work are as follows. Firstly, this method should be applied to larger-scale fuzzy CSPs and real-world CSPs. In real-world CSPs such as timetable planning, there are many mutually contradicting constraints. Some constraints or combinations of specific values are more important and others are less important. This method can probably be applied to such types of problems, because the constraints can probably be regarded as fuzzy constraints. Secondly, there are many parts or parameters of this method which can be improved. For example, parameters such as k and the initial value of the frustrations could be revised.

Acknowledgment

The author thanks Yuzuru Sato from the University of Tokyo for developing the rule shown in Figure 3 and the graphics data and routines for the USA map coloring (Figure 5).

References

[Fre 92] Freuder, E. C., and Wallace, R. J.: Partial Constraint Satisfaction, *Artificial Intelligence*, 58, 21–70, 1992; also: *IJCAI '89*, 278–283, 1989.

[Har 68] Hart, P. E., Nilsson, N. J., and Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Trans. SSC*, SSC-4, 100–107, 1968.

[Kan 92] Kanada, Y.: Toward a Model of Computer-based Self-organizing Systems, *Proc. 33rd Programming Symposium*, 1992 (in Japanese).

[Kan 94a] Kanada, Y., and Hirokawa, M.: Stochastic Problem Solving by Local Computation based on Self-organization Paradigm, *27th Hawaii International Conference on System Sciences,* 82–91, 1994.

[Kan 94b] Kanada, Y.: Methods of Controlling Locality in Problem Solving Using CCM: A Model for Emergent Computation, *SWoPP '94*, 29–38, Information Processing Society of Japan, 1994 (in Japanese).

[Min 92] Minton, S., Johnston, M. D., Philips, A. B., and Laird, P.: Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems, *Artificial Intelligence*, 58, 1–3, 1992.

[Mor 93] Morris, P.: The Breakout Method For Escaping From Local Minima, *11th National Conference on Artificial Intelligence (AAAI '93)*, 40–45, 1993.

[Rut 94] Ruttkay, Z.: Fuzzy Constraint Satisfaction, *3rd IEEE Int. Conf. on Fuzzy Systems*, 1263–1268, 1994.

[Sel 92] Selman, B., Levesque, H., and Mitchell, D.: A New Method for Solving Hard Satisfiability Problems, *10th National Conference on Artificial Intelligence (AAAI '92)*, 440–446, 1993.

[Sel 93] Selman, B., and Kautz, H.: Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems, *13th Int. Joint Conf. on Artificial Intelligence (IJCAI '93)*, 290–295, 1993.

[Tak 92] Takefuji, Y.: *Neural Network Parallel Processing*, Kluwer Academic Publishers, 1992.