Keeping the Stability of Solutions in Dynamic Fuzzy CSPs

Yasuhiro Sudo and Masahito Kurihara
Graduate School of Information Science and Technology
Hokkaido University
Sapporo, Japan
Email: {sudoy, kurihara} @complex.eng.hokudai.ac.jp

Takuto Yanagida
Graduate School of Information Science and Technology
Hokkaido University
Sapporo, Japan
Email: takty@main.ist.hokudai.ac.jp

***Abstract*—A fuzzy constraint satisfaction problem (FCSP) is an extension of the classical CSP, a powerful tool for modeling various problems based on constraints among variables. Meanwhile, a dynamic CSP is a framework for modelling the dynamic transform of problems. These schemes are the technique to formulate real world problems as CSPs, more easily.**

The CSP model that combines these is already has splendid researches. The Fuzzy Local Cange algorithm is practicable enough in small-scale problems. However, when the scale of the problems grows to some degree, it becomes necessary to use approximate methods. Basically, the algorithms for solving CSPs are classified into two categories: the systematic search (complete methods based on search trees) and the local search (approximate methods based on iterative improvement). Both have merits and demerits.

In this paper, we tested a hybrid, approximate method called the SRS algorithm, in case of large-scale problems. SRS repeats spreading and shrinking a set of search trees in order to repair local constraints until the satisfaction degree of the worst constraints (which are the roots of the trees) is improved. In this process, the "stability" of solutions can be maintained because the reassignment is locally limited. We empirically show that SRS keeps the stability of solutions rather than starting the search from scratch. It is able to quickly get a good-quality approximate and stabilized solution of sufficiently large size of problems.

I. Introduction

A *constraint* is a restriction on a space of possibilities; it is a piece of knowledge that narrows the scope of this space. Because constraints arise naturally in most areas of human endeavor, they are the most general means for formulating regularities that govern our computational, physical, biological, and social worlds. Many problems arising in such domains can be naturally modeled as constraint satisfaction problems (CSPs). A CSP consists of a finite set of *variables*, each associated with a finite *domain* of values, and a set of *constraints* among the variables. A *solution* is an assignment of a value to every variable such that all constraints are satisfied.

Since CSPs are NP-complete problems in general, no efficient and complete algorithms for solving CSPs exist and the increase in the worst-case computation time is exponential in the size of the problems. In most of the cases, however, we can obtain a solution in practical time by using incomplete algorithms.

CSP algorithms are usually categorized into two classes. One of them is tree search algorithms based on systematic extension of partial, consistent assignments and backtracking. The other is local state-space search algorithms based on iterative improvement in inconsistent, full assignments. These techniques have been developed almost independently, but recently, much attention has been paid to hybrid methods integrating the virtues of each class of algorithms. An example is the decision-repair algorithm developed in [7], an extended version of a prize-winning paper presented at AAAI-2000.

From the viewpoint of soft computing, however, the traditional CSP is too rigid to formulate real world problems. In order to improve this situation, much work has been done on the extensions of CSPs. The fuzzy CSP (FCSP) is one of such extensions, where constraints are represented by fuzzy relations, which admit incomplete solutions for providing useful information for solving real-world problems[1], [8].

Meanwhile, a dynamic CSP (DCSP) is a framework for modelling the dynamic transform of problems. In many real situations, it is often necessary to re-solve the changed problem. Where, if the searching starts from scratch, the effort to solve a adjacent problem will have been wasted. The key to efficiently solving DCSPs is to re-use the resource such adjacent information as much as possible. The effectiveness in re-use the adjacent solution in the way of default assignments, and min-conflict heuristics are known[2], [3].

Spread-Repair-Shrink (SRS) algorithm is a FCSP solver developed by the authors [5]. The evaluation function of FCSPs is defined as maximizing satisfaction degree of the most violated constraint. The SRS algorithm boosts up searching efficiency by iterative improvement that is restricted around the most violated constraint. This approach will contributes to keep stability of solutions in DCSPs. Based on comprehensive experiments by using randomly-generated problems with various sets of parameter settings, we show that SRS is more effective than starting the search from scratch, in sufficiently large size of problems.

This paper is organized as follows. In Section 2, we briefly review the traditional CSP , the fuzzy CSP and the dynamic CSP. In Section 3, we present the SRS algorithms. In Section 4, we show the experimental results and discuss the performance of the algorithm. Section 5 contains a summary of our work.

II. Fuzzy, Dynamic CSP

A. Classical CSP

A CSP is defined by a set of variables $X = \{x_i\}_{i=1}^n$ that take values from finite domains $D = \{D_i\}_{i=1}^n$ under a set of constraints $C = \{c_k\}_{k=1}^r$, where c_k denotes a relation R_k on a subset $S_k (S_k \subseteq X)$ of X.

$$R_k \subseteq D_{k_1} \times \cdots \times D_{k_w} \ for \ S_k = \{x_{k_1}, \cdots, x_{k_w}\}$$

S_k is called the *scope* of R_k. If $w = 1, 2,$ or 3, then the relation is called a *unary, binary,* or *ternary* relation, respectively.

B. Fuzzy CSP

A Fuzzy CSP(FCSP) is defined as an extension of classical CSP where constraints c_k are represented by fuzzy relations R_k with their membership functions defined by

$$\mu R_k : \prod_{x_i \in S_k} D_i \rightarrow [0, 1] \qquad \cdots (1)$$

In other words, a membership value $\mu R_k(v_{S_k})$ of an assignment v_{S_k} to the variables in the scope S_k of a constraint C_k takes $[0, 1]$. The membership value is also called the *degree of satisfaction*.

The fuzzy conjunction (AND) $C_k \wedge C_l$ of two constraints C_k and C_l is the fuzzy relation $R_k \cap R_l$ with scope $S_k \cup S_l$ and membership function

$$\mu R_k \cap R_l(v) = min(\mu R_k(v[S_k]), \mu R_l(v[S_l])) \qquad \cdots (2)$$

where $v[S]$ is the *projection*. Since the FCSP requires the satisfaction of the fuzzy conjunction of all the fuzzy constraints, the degree of satisfaction of the whole FCSP is defined as the minimum degree of satisfaction as follows.

$$\mu \bigcap_{k=1}^{r} R_k(v) = \min_{1 \le k \le r} (\mu R_k(v[S_k])) \qquad \cdots (3)$$

In order to simplify the notation, we denote the minimum degree of satisfaction of the whole FCSP, given v, by C_{min}:

$$C_{min}(v) = \min_{1 \le k \le r} (\mu R_k(v[S_k])) \qquad \cdots (4)$$

Given v, any constraint c_k that gives this minimum is denoted by c^* and called a *most violated constraint*. If $C_{min}(v) > 0$, v is called a *solution* of the FCSP. A solution that maximizes $C_{min}(v)$ (i.e., the degree of satisfaction of c^*) is called an *optimal solution*. Therefore, a FCSP is regarded as an optimization problem which requires the search for an assignment v that maximizes the minimum degree of satisfaction of the constraints. Thus the optimal value of the objective function (4) is formulated as follows.

$$\max_{v}(\min_{1 \le k \le r} (\mu R_k(v[S_k]))) \qquad \cdots (5)$$

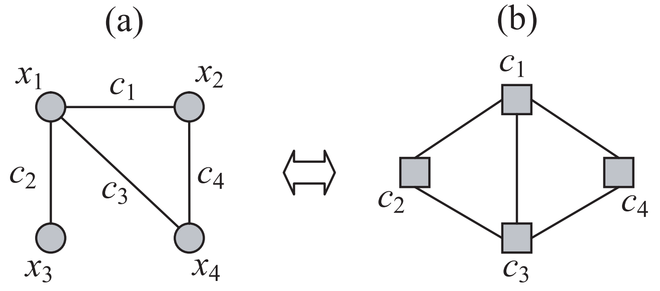

Fig. 1. Constraint Graph and Dual Constraint Graph

C. Constraint Networks

The structure of CSPs and FCSPs can be represented by a *constraint graph* (Fig.1a). Nodes and edges of the graph are corresponding to variables and constraints. If the constraint c_k is binary, the two nodes in the scope S_k is connected by an edge. If the constraint is ternary or of higher order, the graph is a hypergraph that has hyperedges for connecting 3 or more nodes in its scope. For such cases, it is convenient to use a *dual constraint graph* (Fig.1b), which represents constraints as nodes and sets of variables as edges; an edge is drawn between two nodes corresponding to constraints c_i and c_j when $S_i \cap S_j \neq \phi$. Note that the edges are ordinary (not hyper-) edges connecting two nodes. Thus dual constraint graphs are ordinary graphs, not hypergraphs.

D. Dynamic CSP

A DCSP is defined as follows[3]:

$$DC = \{CSP_0, CSP_1, CSP_2, \cdots\} \qquad \cdots (6)$$

CSP_i is a new problem such that revised CSP_{i-1}. The model that combined fuzzy and dynamic CSP was proposed as a sequence of FCSPs in the literature [2]. In this paper, we define a DFCSP as a tupple of two FCSPs to simplify the model.

$$FDC = \{FCSP_P, FCSP_F\} \qquad \cdots (7)$$

Solving DFCSP is to search optimum assignment of $FCSP_F$ that is changed $FCSP_P$. In the case of real world applications, the difference between both assignment is desired as small as possible. We define the degree of similarity $\delta(v, v')$ as follows:

$$\delta(v, v') \ = \ \frac{\sum_{i=1}^{n}}{n} \begin{cases} 1 & v(i) = v'(i) \\ 0 & otherwise \end{cases} \qquad \cdots (8)$$

Where $v(i)$ is an assignment of a variable x_i. The larger $\delta(v, v')$ is, the more stable solution is. Such "stability" is recommended in a DCSP framework[3]. However, in this paper, it is taboo to deteriorate a quality of solution (increase constraint violations) in order that boost up the stability. Therefore, the formula (3) is confined to the sub-evaluation.

III. The SRS Algorithm

If the domain of CSP and FCSP is a finite and discrete set, it is possible to generate a complete search tree to exhaustively search for solutions. However, the time complexity increases

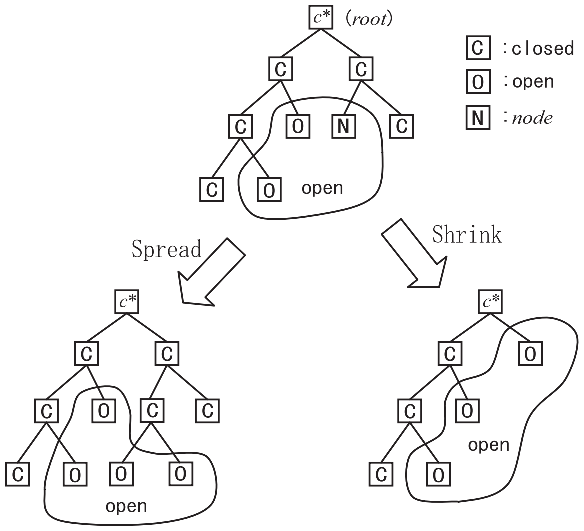

Fig. 2. Spread and Shrink

exponentially with the size of the problems in the worst case. Basically, the algorithms for solving CSPs are classified into two categories: the systematic search (complete methods based on search trees) and the local search (approximate methods based on iterative improvement). Both have merits and demerits. Recently, much attention has been paid to hybrid methods for integrating both merits to solve CSPs efficiently, but almost no attempt has been made so far for solving FCSPs.

In this section, we present the Spread-Repair-Shrink(SRS) algorithm, which will turn out to be an efficient procedure to obtain a good-quality approximate solution to FCSPs. The type of hybridizing is summarized as follows:

- performing an overall local search, and using systematic search in order to control constraint propagation for escaping from local optima.

1) : The Basic Structure: As discussed in the previous section, the aim of FCSPs is to maximize the degree of satisfaction of the most violated constraints (c^*'s). To solve the problems based on the local search framework, the Spread-Repair (SR) algorithm developed by the authors in [4] tries to *repair* a target constraint c^* repetitively by changing the value of a variable in its scope. When trapped in a local maximum, SR tries to escape from the state by the operation called *spreading*, which changes the target of the repair to the ‘neighbor’ constraints surrounding the current target.

The SRS algorithm improves SR by combining the original local search framework with a new systematic search operation called *shrinking*, which is almost an inverse operation of spreading because it changes the target back to the previous target in order to propagate the effect of the repair to c^*. By this process, the redundancy of the computation observed in SR is effectively suppressed.

More precisely, given a set of most violated constraints, SRS maintains a forest (i.e., a set of trees) consisting of search trees which are subgraphs of the dual constraint network. Each root node corresponds to one of the most violated constraints, and the leaf nodes of the trees correspond to the potential target constraints which are open to be checked for the repair. By spreading, repairing, and shrinking the trees, the algorithm tries to repair all the c^*'s effectively. The structure of the algorithm, based on the open/closed-lists algorithm well-known in the AI literature, consists of the three modes *Repair*, *Spread*, and *Shrink*, to be switched from one to another in the running time.

Many other implementations of Repair operation may exist. It depends on the problem domains and the purposes of the application which implementation we should use. In the experiments in Section 4, we have used SRS3[5], which is superior to others in terms of CPU time, the quality of solutions, and implementation cost.

We show the SRS algorithms in Fig.3. Here are some comments in the following.

- The main program computes and passes the set of the most violated (the worst) constraints, C, to the function SRS. This process is repeated until SRS returns FALSE.
- Function SRS(C) controls the search tree by appropriately calling procedure Spread(), Repair($node$), or Shrink() based on the scheme described in this section. If all elements of C were repaired, this function returns TRUE; otherwise, it returns FALSE.
- Procedure Spread() expands the search tree.
- Procedure Shrink() shrinks the search tree recursively.
- Function Repair(c) tries to repair constraint c. If c has been repaired, the function returns TRUE; otherwise, it returns FALSE.

A. Keeping stabilities

The structure of the spreading process of the SRS algorithm is similar to that of the graph-search procedure well-known in the AI literature. Corresponding to the goals (or targets) in the graph-search are the nodes which can be locally repaired by changing a value of a variable in SRS. When the most violated constraints c^* cannot be repaired, SRS tries to find such targets by constructing the search trees rooted at c^*. This process, called *expansion* in the graph-search, is called *spreading* in SRS.

The open and closed lists are maintained in the process. In each loop, the procedure takes a node out of the open list as a candidate for Repair operation. If it cannot be repaired, it is put into the closed list. Then, the nodes adjacent to that node are put into the open list as its children whenever they are not still contained in the open list nor the closed list (Fig.2).

When a target constraint (node) has been found and repaired, SRS changes its mode Shrink, in which SRS tries to propagate the effects of the repair back to the root c^*. For this purpose, its focus is moved from the repaired node to its parent node, which is closer to the root, and all the descendants of the new focus are removed from the open and closed lists (Fig.2).

```
Function SRS(constraints C)
    {input C: a set of the worst constraints}
    closed ← $\phi$.
    open ← C.
    While C ≠ $\phi$ and open ≠ $\phi$ do
        node ← the first element of open.
        open ← open \ node.
        If not Repair(node) then
            Spread()
        Else if node ∈ C then
            C ← C \ {node}.
        Else if Repair(the parent of node) then
            Shrink( )
        Else
            Spread( )
        End if
        Sort open by heuristic h( ).
    End while
    If C = $\phi$ then return TRUE
    Else return FALSE
End function

Procedure Spread( )
    closed ← closed ∪ node.
    open ← open ∪ neighbors(node) \ closed.
End procedure

Procedure Shrink( )
    node ← parent of node.
    open ← open \ the descendants of node.
    closed ← closed \ descendants of node.
    If node ∈ C then
        C ← C \ {node}.
    Else
        open ← open ∪ node.
        If Repair(the parent of node) then Shrink( )
    End if
End procedure

Function Repair(constraint c)
    Try to repair c.
    If repaired then
        return TRUE
    Else
        return FALSE
    End if
End function

Begin program
    Do while SRS(worst constraints).
End program
```

Fig. 3. Spread-Repair-Shrink Algorithm

Then this shrinking process is repeated recursively until the focus reaches the root or it turns out that the focus cannot be repaired any more. Note that SRS performs efficiently by maintaining the path from c^* to the repaired node in the search tree, while our previous algorithm SR [4] has to restart the spreading from the scratch after each repair of the target.

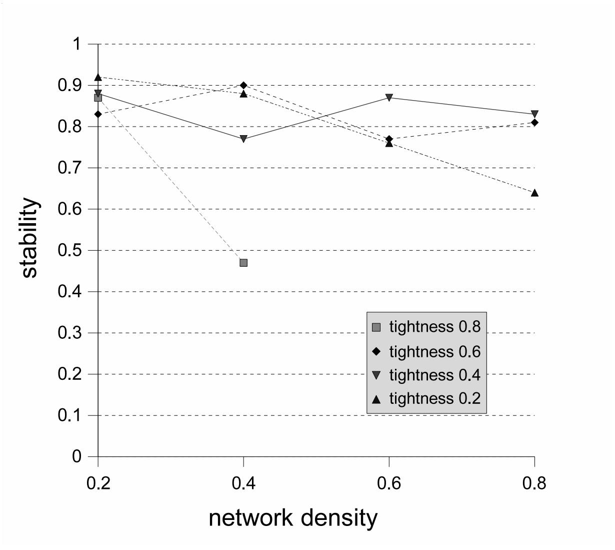

Fig. 4. 15 variables, 10 domain's size and 1 constraint replaced

Thus the SRS algorithm boosts up searching efficiency by iterative improvement that is restricted around the most violated constraint. This means that the changing the value of each variable is suppressible. As a result, this approach will contributes to keep stability of solutions in DCSP schemes.

IV. Evaluation

We have tested the performance of the SRS algorithm by the performance is measured by a comprehensive experiment based on randomly-generated problems with various values for the density and tightness parameters. The performance is measured by the stability (formula 8).

The test problems are randomly generated binary FCSPs with parameters n, Δ, d, and t, where n is the number of variables, Δ is the common size of the domains $D_1, \ldots, D_n$, d is the density of the constraint network, and t is the average tightness of constraints[13]. In this section, we define d and t as follows.

$$d = \frac{r}{(n^2 - n)/2} \qquad \cdots (9)$$

$$t = 1 - \frac{1}{r} \sum_{k=1}^{r} \left[\frac{1}{\prod_{i=1}^{w} |D_i|} \cdot \sum_{v \in \prod_{i=1}^{w} D_{k_i}} \mu R_k(v) \right] \qquad \cdots (10)$$

Here, $S_k = \{x_{k_1}, \cdots, x_{k_w}\}$. In the experiment, we set $n = 15, 30, 50$, $\Delta = 10$ and d and t are chosen from the set $\{0.2, 0.4, 0.6, 0.8\}$. For each combination of sixteen (d, t) values, 100 problem instances have been randomly generated and solved.

The simulation is divided in 4 steps as follows:

- get a approximate solution by using SRS method.
- select a constraint(s), and make a change so that the satisfaction degree falls.

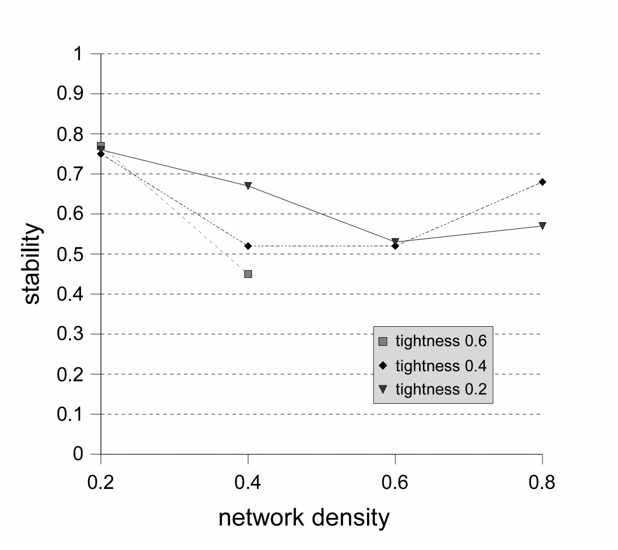

Fig. 5. 30 variables, 10 domain's size and 3 constraints replaced

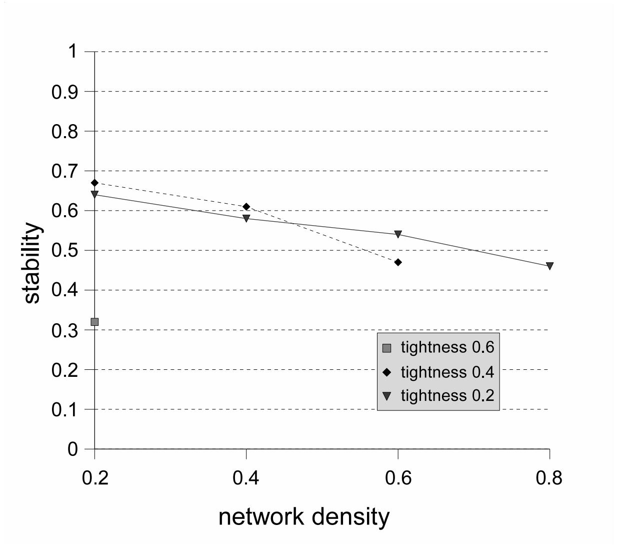

Fig. 6. 50 variables, 10 domain's size and 5 constraints replaced

- apply the SRS method again, using the adjacent solution in the way of default assignments.
- check the stability between both solutions.

The results are shown in fig.4, fig.5 and fig.6. The average objective values are summarized in the figure. The constraint was replaced according to the scale of the problem as 1, 3, and 5.

When the problem becomes large-scale, stability also shows the tendency to fall. However, in almost all instances, the degree of stability exceeds 50%. From these results, we can obtain a practical stable solution by using SRS algorithm.

V. Conclusion

We have tested the Spread-Repair-Shrink algorithm for obtaining stable approximate solutions to Dynamic Fuzzy Constraint Satisfaction Problems. The algorithm can be thought of a new type of hybrid algorithm for combining systematic search with local search, because SRS exploits the MAX-MIN feature of (D)FCSP in focusing on the most violated constraint (c^*) to be repaired efficiently by systematic Spread, Repair, and Shrink operations on the search trees. The experimental results show that SRS is useful when we want a relatively good stable approximate solution for large-scale dynamic problems.

Future research topics include further improvement of SRS in terms of quality and stable degree. Also interesting is an extension of the idea to broader classes of soft computing such as the ones called Soft Constraint Satisfaction Problems (Max CSPs, Weighted CSPs, etc.) [14], [15].

References

[1] Zs. Ruttkay, "Fuzzy constraint satisfaction", *Proceedngs of 3rd IEEE Intern. Conf. on Fuzzy Systems*, Vol. 3, pp.1263-1268, 1994.

[2] I. Miguel and Q. Shen, "Fuzzy *rr*DFCSP and Planning", *Artificial Intelligence* Vol. 148, pp.11-52, 2003.

[3] G. Verfaillie and T. Schiex, "Solution Reuse in Dynamic Constraint Satisfaction Problems", *Proceedings of the 12th National Conf. on Artificial Intelligence*, pp.307-312, 1994.

[4] Y. Sudo, M. Kurihara and T. Mitamura, "Spread-Repair Algorithm for Solving Extended Fuzzy Constraint Statisfaction Problems", *Proceedings of the 4th IEEE International Workshop on Soft Computing as Transdisciplinary Science and Technology*, pp.891-901, 2005.

[5] Y. Sudo and M. Kurihara "Spread-Repair-Shrink: A Hybrid Algorithm for Solving Fuzzy Constraint Satisfaction Problems", *Proceedings of the IEEE World Congress on Computational Intelligence*(WCCI2006), pp.9969-9975, 2006.

[6] S. Minton, M.D. Johnston, A.B. Philips and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems", *Artificial Intelligence* Vol. 58, pp.161-205, 1992.

[7] N. Jussien and O. Lhomme, "Local search with constraint propagation and conflict-based heuristics", *Artificial Intelligence*, Vol. 139, pp.21-45, 2002

[8] P. Meseguer and J. Larrosa, "Solving fuzzy constraint satisfaction problems", *Proceedings of 6th IEEE Intern. Conf. on Fuzzy Systems*, Vol. 3, pp.1233-1238, 1997

[9] R.M. Haralic, and G.L. Elliot, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems", *Artificial Intelligence*, Vol. 14, pp.263-313, 1980

[10] H.M. Adorf and M.D. Johnston, "A discrete stochastic neural networks algorithm for constraint satisfaction problems", *Proceedings International Joint Conference on Newral Networks*, CA, 1990

[11] J.H.Y. Wong and H. Leung, "Extending GENET to Solve Fuzzy Constraint Satisfaction Problems", *Proceedngs of 15th National Conf. on Artifitial Intelligence*(*AAAI*-98), pp.380-385, 1998

[12] J. Bowen, and G. Dozier, "Solving Randomly Generated Fuzzy Constraint Networks Using Evolutionary/Systematic Hill-Climbing", *Proceedings of 5th IEEE Intern. Conf. on Fuzzy Systems*, vol. 1, pp.226-231, 1996

[13] J. Culberson and T. Walsh, "Tightness of Constraint Satisfaction Problems", APES-29, 2001

[14] R.J. Wallace, "Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problems", *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pp.188-195, 1996

[15] S. Bistarelli, "Semirings for Soft Constraint Solving and Programming" (Lecture Notes in Computer Science 2962), SpringerVerlag, 2004

[16] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems", *Artificial Intelligence* vol. 58, pp.161-205, 1992