Keeping the Stability of Solutions to 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 is an extension of the classical CSP, a powerful tool for modeling various problems based on constraints among variables, and a dynamic CSP is a framework for modelling the transformation 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 Change algorithm is practicable enough in small-scale problems, but larger problems require the use of approximate methods. The algorithms for solving CSPs are classified into two categories: systematic searches (complete methods based on search trees), and local searches (approximate methods based on iterative improvement). Both have advantages and disadvantages.**

In the work reported in this paper we tested a hybrid approximate method, called the Spread-Repair-Shrink algorithm, on dynamic, large-scale problems. The algorithm repairs local constraints by repeatedly spreading and shrinking a set of search trees until the degree to which the worst constraints (the roots of the trees) are satisfied is improved. In this process, the "stability" of solutions can be maintained because the reassignment is locally limited. Additionaly, we innovate SRSD filter as after filtering. We empirically show that Spread-Repair-Shrink and SRSD algorithm keep the stability of solutions rather than other algorithms. It is able to quickly get a good-quality approximate and stabile solution to a large problem.

I. Introduction

A *constraint* is a restriction on a space of possibilities, 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 increases exponentially with 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 much attention has recently been paid to hybrid methods combining the strengths 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 classical CSP is too rigid to formulate real-world problems. Much work has therefore been done on the extensions of CSPs. In a fuzzy CSP (FCSP), for example constraints are represented by fuzzy relations, which accommodate incomplete solutions providing information useful for solving real-world problems[1], [8]. And a dynamic CSP (DCSP) is a framework for modelling the dynamic transform of problems. In many situations a problem changes and it is necessary to solve the changed problem. If the search for a solution to the changed problem starts from scratch, the effort that had been expended to solve the original problem will have been wasted. The key to efficiently solving DCSPs is to re-use resources such as 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].

The Spread-Repair-Shrink (SRS) algorithm is a FCSP solver developed by the authors [5]. The evaluation function of FCSPs is defined as maximizing the degree to which the most violated constraint is satisfied. The SRS algorithm boosts search efficiency by restricting iterative improvement to the most violated constraint. This will contribute to the stability of solutions in DCSPs. Here we show the results of comprehensive experiments, using randomly-generated problems with various sets of parameter settings, that when the problem is sufficiently large the SRS is more effective than starting each search from scratch.

This paper is organized as follows. In Section 2, we briefly review the classical CSP, the fuzzy CSP, and the dynamic CSP. In Section 3, we describe the SRS algorithms and SRSD filter. In Section 4, we show the experimental results and discuss the performance of the algorithm. In Section 5, we conclude the paper by summarizing it and indicating directions for future work.

II. Classical, Fuzzy, and Dynamic CSPs

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, the relation is called a *unary, binary,* or *ternary* relation, respectively.

B. Fuzzy CSP

A fuzzy CSP (FCSP) is defined as an extension of the classical CSP. In an FCSP the constraints c_k are represented by fuzzy relations R_k whose membership functions are defined by

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

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 fuzzy conjunction of all the fuzzy constraints to be satisfied, the degree of satisfaction of the whole FCSP is defined as the minimum degree of satisfaction as follows.

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

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

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

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*. An FCSP is therefore 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.

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

Fuzzy GENET (FGENET) is a neural network model, for solving binary FCSPs. Corresponding to each variable of a given FCSP is a cluster of neurons which represent the values in the domain of the variable. A pair of neurons corresponding to two values forbidden by the constraints are connected to each other, given a negative weight for suppressing firing each other. The states of the neurons are changed asynchronously according to inputs. When trapped in local maxima, the procedure tries to escape from them by increasing weights of appropriate neurons responsible for the trap. By this simple mechanism, the method successfully attained almost the same good performance as the evolutionary/systematic hill-climbing method.

C. Dynamic CSP

A dynamic CSP (DCSP) is defined as follows[3]:

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

CSP_i is a new problem revised such that CSP_{i-1}. The model that combined fuzzy and dynamic CSPs as a sequence of FCSPs has been proposed[2], but in this paper we simplify that model by defining a DFCSP as a tupple of two FCSPs

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

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

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

Where $v(i)$ is an assignment of a variable x_i. The larger $\delta(v, v')$ is, the more stable the solution is. Such stability is recommended in a DCSP framework[3]. Here, however, we do not degrade the quality of a solution (increase constraint violations) in order to increase its stability. The formula (3) is therefore confined to the sub-evaluation.

In this numerical model, the Fuzzy Local Change (FLC) algorithm is well known. However, FLC is categorize d in a systematic search, so it is inefficient in large scale problems.

III. SRS Algorithm

If the domain of a CSP or FCSP is a finite and discrete set, it is possible to generate a complete search tree and search for solutions exhaustively, but in the worst case the time complexity increases exponentially with the size of the problem. Although extensive efforts have been devoted to developing hybrid methods combining the advantages of the systematic and local search methods for solving CSPs, methods for solving FCSPs have received almost no attention.

In this section we describe a spread-repair-shrink (SRS) algorithm that efficiently obtains good 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.

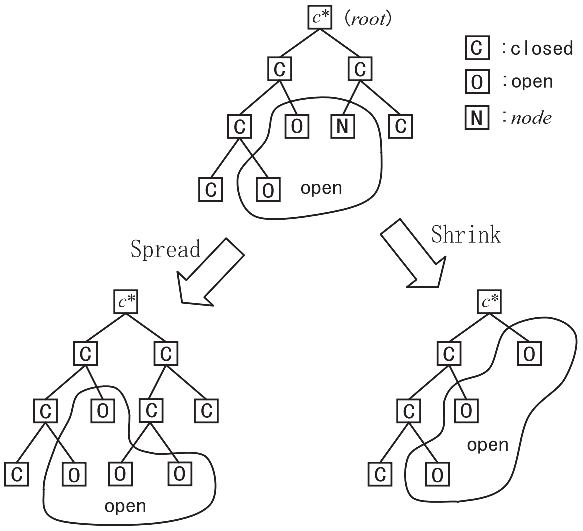

Fig. 1. Spread and Shrink operations

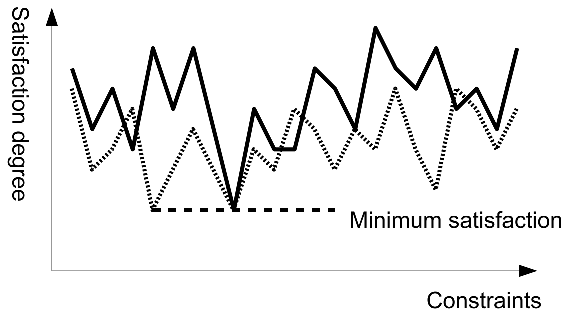

Fig. 2. The evaluation is decided by minimum satisfactions

```
Procedure SRSD( )
    for each $x \in X$
        if $\mu R(v'[x]) \geqq C_{min}(v[x])$
            $v[x] \leftarrow v'[x]$
            SRSD( )
        End if
End procedure
```

Fig. 3. SRSD filter algorithm

1) : As explained in the previous section, FCSPs are solved by maximizing the degree to which the most violated constraints (c^*'s) are satisfied. The Spread-Repair (SR) algorithm developed by the authors in [4] repeatedly tries to *repair* a target constraint c^* by changing the value of a variable in its scope. When trapped in a local maximum, it tries to escape by the operation called *spreading*, which changes the target of the repair to the 'neighbor' constraints surrounding the current target.

The SRS algorithm combines the original local search framework of the SR algorithm 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^*. This process reduces the computational redundancy that occurs with the SR algorithm.

More precisely, given a set of most violated constraints, the SRS algorithm maintains a forest (i.e., a set of trees) consisting of search trees that 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 that can 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 the repair operation could be used, and which one we should use depends on the problem domains and the purposes of the application. In the experiments described in Section 4 we used SRS3[5], which is the best in terms of CPU time, quality of solutions, and implementation cost.

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 the value of a variable. When the most violated constraints c^* cannot be repaired, the algorithm 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 the SRS algorithm.

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. The nodes adjacent to that node are then put into the open list as its children whenever they are not already in the open list or the closed list (Fig. 1).

When a target constraint (node) has been found and repaired, the SRS algorithm changes its mode Shrink, in which it 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. 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

TABLE I
15 VARIABLES, 10 DOMAIN'S SIZE AND 5 PERCENT CONSTRAINTS REPLACED

		t =0.2		t =0.4		t =0.6		t =0.8	
	d	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$
FLC	d =0.2	1.00	0.47	0.80	0.50	0.59	0.49	0.42	0.46
FLC+SRSD			0.49		0.52		0.51		0.49
SRS3		0.99	0.77	0.78	0.80	0.53	0.79	0.37	0.77
SRS3+SRSD			0.86		0.89		0.88		0.85
FGN		1.00	0.28	0.80	0.30	0.54	0.29	0.37	0.27
FGN+SRSD			0.44		0.45		0.46		0.45
FLC	d =0.4	0.89	0.37	0.63	0.35	0.40	0.36	0.15	0.37
FLC+SRSD			0.40		0.38		0.38		0.41
SRS3		0.84	0.64	0.57	0.61	0.34	0.62	0.09	0.62
SRS3+SRSD			0.73		0.73		0.71		0.75
FGN		0.89	0.22	0.58	0.19	0.34	0.20	0.08	0.20
FGN+SRSD			0.27		0.25		0.26		0.26
FLC	d =0.6	0.79	0.22	0.52	0.20	0.32	0.21	0.00	1.00
FLC+SRSD			0.24		0.24		0.22		1.00
SRS3		0.75	0.55	0.46	0.53	0.25	0.51	0.00	1.00
SRS3+SRSD			0.66		0.65		0.63		1.00
FGN		0.76	0.19	0.46	0.17	0.26	0.17	0.00	1.00
FGN+SRSD			0.22		0.21		0.23		1.00
FLC	d =0.8	0.74	0.22	0.46	0.20	0.26	0.21	0.00	1.00
FLC+SRSD			0.23		0.21		0.22		1.00
SRS3		0.69	0.40	0.42	0.42	0.21	0.41	0.00	1.00
SRS3+SRSD			0.50		0.52		0.52		1.00
FGN		0.68	0.14	0.39	0.12	0.20	0.12	0.00	1.00
FGN+SRSD			0.21		0.19		0.21		1.00

TABLE II
15 VARIABLES, 10 DOMAIN'S SIZE AND 10 PERCENT CONSTRAINTS REPLACED

		t =0.2		t =0.4		t =0.6		t =0.8	
	d	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$
FLC	d =0.2	1.00	0.49	0.79	0.49	0.58	0.48	0.40	0.51
FLC+SRSD			0.54		0.52		0.52		0.52
SRS3		0.96	0.67	0.75	0.68	0.53	0.67	0.36	0.69
SRS3+SRSD			0.78		0.80		0.78		0.80
FGN		1.00	0.26	0.77	0.25	0.54	0.26	0.36	0.24
FGN+SRSD			0.40		0.39		0.39		0.39
FLC	d =0.4	0.88	0.25	0.62	0.26	0.41	0.26	0.09	0.25
FLC+SRSD			0.28		0.28		0.29		0.27
SRS3		0.84	0.49	0.57	0.50	0.35	0.51	0.00	1.00
SRS3+SRSD			0.56		0.56		0.58		1.00
FGN		0.87	0.15	0.57	0.14	0.34	0.14	0.00	1.00
FGN+SRSD			0.18		0.17		0.18		1.00
FLC	d =0.6	0.79	0.15	0.52	0.15	0.31	0.14	0.00	1.00
FLC+SRSD			0.19		0.17		0.18		1.00
SRS3		0.76	0.40	0.45	0.42	0.26	0.39	0.00	1.00
SRS3+SRSD			0.48		0.49		0.48		1.00
FGN		0.76	0.12	0.45	0.13	0.27	0.11	0.00	1.00
FGN+SRSD			0.16		0.18		0.17		1.00
FLC	d =0.8	0.73	0.13	0.47	0.13	0.26	0.11	0.00	1.00
FLC+SRSD			0.15		0.14		0.13		1.00
SRS3		0.67	0.32	0.40	0.34	0.19	0.33	0.00	1.00
SRS3+SRSD			0.41		0.42		0.40		1.00
FGN		0.69	0.11	0.39	0.12	0.18	0.12	0.00	1.00
FGN+SRSD			0.19		0.18		0.19		1.00

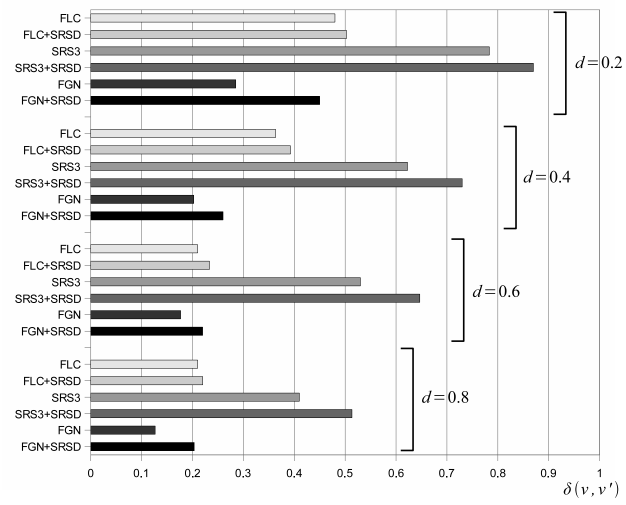

Fig. 4. Stability of algorithms

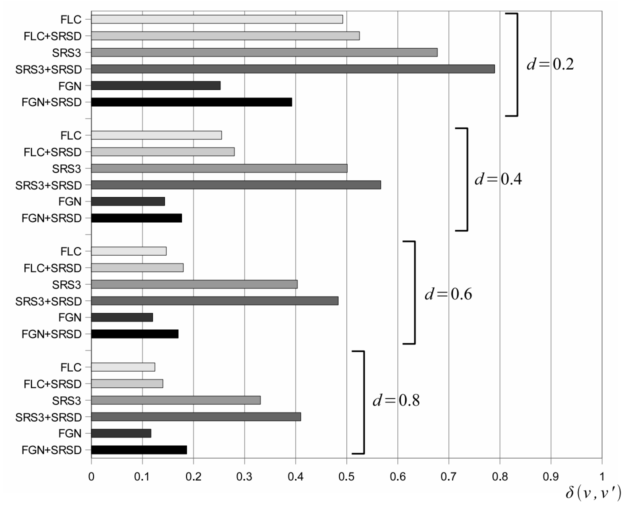

Fig. 5. Stability of algorithms

restart the spreading from the scratch after each repair of the target.

The SRS algorithm thus increases the efficiency of its searching 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 contribute to keep stability of solutions in DCSP schemes.

B. SRSD Filter

Now, if an effectiveness of a solution of a DFCSP is given by the worst violated constraint C_{omega}, we can up the stability under the restriction that is defined $\mu R(v'[x]) \geq C_{min}(v[x])$ (Fig.2). Here, v' is former assignment. So, we compose a SRSD method that return assignments greedy (Fig.3). However, this method vitiates average of satisfaction degree that is defined as follows.

$$C_{ave} = \frac{1}{r}\sum_{k=1}^{r} C_k \qquad \cdots (9)$$

IV. Experimental Evaluation

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

The test problems were 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

TABLE III
30 VARIABLES, 10 DOMAIN'S SIZE AND 3 PERCENT CONSTRAINTS REPLACED

		t =0.2		t =0.4		t =0.6		t =0.8	
	d	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$
SRS3	d =0.2	0.82	0.53	0.52	0.55	0.27	0.52	0.00	1.00
SRS3+SRSD			0.69		0.71		0.70		1.00
FGN		0.84	0.22	0.49	0.20	0.23	0.21	0.00	1.00
FGN+SRSD			0.40		0.38		0.39		1.00
SRS3	d =0.4	0.64	0.35	0.38	0.38	0.18	0.34	0.00	1.00
SRS3+SRSD			0.51		0.52		0.50		1.00
FGN		0.62	0.17	0.27	0.16	0.07	0.16	0.00	1.00
FGN+SRSD			0.36		0.38		0.37		1.00
SRS3	d =0.6	0.59	0.30	0.32	0.29	0.10	0.29	0.00	1.00
SRS3+SRSD			0.41		0.40		0.41		1.00
FGN		0.52	0.16	0.23	0.15	0.09	0.16	0.00	1.00
FGN+SRSD			0.35		0.35		0.36		1.00
SRS3	d =0.8	0.53	0.28	0.26	0.26	0.00	1.00	0.00	1.00
SRS3+SRSD			0.33		0.32		1.00		1.00
FGN		0.46	0.12	0.22	0.13	0.00	1.00	0.00	1.00
FGN+SRSD			0.29		0.30		1.00		1.00

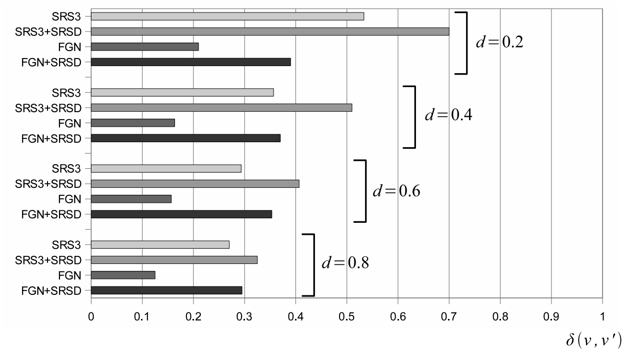

Fig. 6. Stability of algorithms

tightness of constraints[13]. In this section, we define d and t as follows.

$$d = \frac{r}{(n^2 - n)/2} \qquad \cdots (10)$$

$$t = 1 - \frac{1}{r} \sum_{k=1}^{r} \left[\frac{1}{\prod_{i=1}^{w} |D_i|} \cdot \sum_{v \in \prod_{i=1}^{w} D_{k_i}} \mu R_k(v) \right] \qquad \cdots (11)$$

Here, $S_k = \{x_{k_1}, \cdots, x_{k_w}\}$. In the experiment, we set $n = 15, 30$, $\Delta = 10$ 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 were been generated randomly and solved.

The simulation was divided into the following four steps:

TABLE IV
30 VARIABLES, 10 DOMAIN'S SIZE AND 5 PERCENT CONSTRAINTS REPLACED

		t =0.2		t =0.4		t =0.6		t =0.8	
	d	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$	*Cmin*	$\delta(v,v')$
SRS3	d =0.2	0.80	0.48	0.52	0.47	0.27	0.47	0.00	1.00
SRS3+SRSD			0.64		0.63		0.64		1.00
FGN		0.83	0.20	0.48	0.21	0.21	0.20	0.00	1.00
FGN+SRSD			0.37		0.38		0.37		1.00
SRS3	d =0.4	0.63	0.32	0.36	0.33	0.17	0.32	0.00	1.00
SRS3+SRSD			0.45		0.44		0.44		1.00
FGN		0.62	0.16	0.27	0.16	0.06	0.16	0.00	1.00
FGN+SRSD			0.36		0.36		0.36		1.00
SRS3	d =0.6	0.59	0.27	0.33	0.28	0.09	0.27	0.00	1.00
SRS3+SRSD			0.39		0.40		0.38		1.00
FGN		0.53	0.16	0.22	0.15	0.09	0.15	0.00	1.00
FGN+SRSD			0.34		0.34		0.33		1.00
SRS3	d =0.8	0.53	0.27	0.27	0.27	0.00	1.00	0.00	1.00
SRS3+SRSD			0.31		0.32		1.00		1.00
FGN		0.45	0.12	0.21	0.11	0.00	1.00	0.00	1.00
FGN+SRSD			0.29		0.28		1.00		1.00

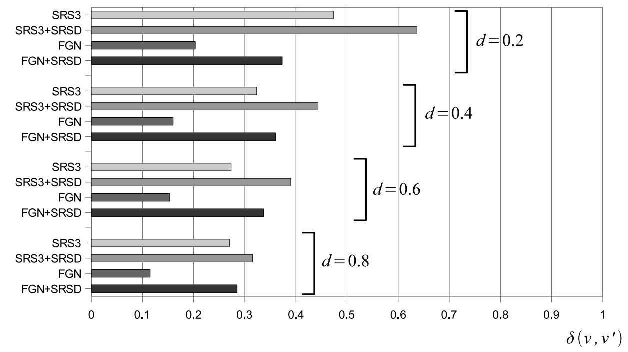

Fig. 7. Stability of algorithms

- get an (approximate) solution by using FLC or SRS algorithm.
- select a constraint(s), and make a change so that the satisfaction degree falls.
- apply the FCSP algorithm again, using the adjacent solution in the way of default assignments.
- apply the SRSD filter.
- check the stability between both solutions.

The results are shown in Tab.1, 2, 3 and 4. The Fig.4, 5, 6 and 7 are graphs expected to be going to extract only stability easily.

The average objective values are summarized in the figures. The constraint was replaced according to the scale of the problem as 3, 5 and 10 Percents.

In all parameter, the results of SRS3 and SRSD is effective. The efect of SRSD after FLC is little, but in other case SRSD heightens stability of the solution. From these results, we see that a practically stable solution can be obtained by using SRS and SRSD algorithm.

V. Conclusion

We have tested the Spread-Repair-Shrink algorithm and SRSD filter for obtaining stable approximate solutions to Dynamic Fuzzy Constraint Satisfaction Problems. SRS and SRSD can be thought of a new type of hybrid algorithm combining systematic search with local search because it 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 the SRS and SRSD are useful when we want relatively good stable approximate solutions to large-scale dynamic problems.

Future research topics include further improvements of solution quality and stability. Also interesting is the extension of the idea to broader classes of soft computing, such as those 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