https://doi.org/10.20965/jaciii.2010.p0425

Paper:

A Tool for Visualizing the Behavior of Fuzzy Constraint Satisfaction Solvers

Takuto Yanagida, Masahito Kurihara, and Hidetoshi Nonaka

Graduate School of Information Science and Technology, Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan
E-mail: {takty, nonaka}@main.ist.hokudai.ac.jp, kurihara@ist.hokudai.ac.jp
[Received December 2, 2009; accepted February 10, 2010]

This paper presents a software tool to intuitively comprehend and analyze fuzzy constraint satisfaction problems (FCSPs) through effective visualization with animation. FCSPs are an extension of constraint satisfaction problems (CSPs), which are used for formulating problems in real-world information systems. Once we formulate a problem as an FCSP, it can be solved (in principle) by any existing general-purpose FCSP solvers developed so far. However, the formulation is sometimes difficult because it requires high abstraction of real-world problems and affects the performance of the solvers. The authors believe that the tool will improve this situation by increasing the visibility of the behavior of the solvers.

Keywords: fuzzy constraint satisfaction problems, visualization, development tool

1. Introduction

Fuzzy constraint satisfaction problems (FCSPs) [1] are an extension of the classical (crisp) constraint satisfaction problem (CSP), a simple model for formulating problems in the real-world information systems studied in the field of computational intelligence and intelligent informatics. FCSPs consist of variables, domains for each variable, and constraints among the variables. Representing constraints by fuzzy relations, they provide suboptimal but useful solutions for a lot of problems that are hard to handle with the classical CSPs.

Once we formulate a problem as an FCSP, it can be solved (in principle) by any existing general-purpose FCSP solvers developed so far; however, the formulation is sometimes difficult because it requires high abstraction of real-world problems and affects the performance of the solvers. That is, the formulation is a problem left to humans. For example, we formulated the layout problem of graphical user interfaces as FCSPs [2, 3], but it was never a trivial formulation.

We believe that the situation of formulations will be improved by increasing the visibility of the behavior of the solvers in order to adjust models once developed. When we utilized the FCSPs as a framework for solving problems, we were often confused because we were not able to understand the behavior of the solvers: e.g., why the solver outputs no solution, how often that happens, and what is the performance bottleneck of the modeled problem. We can see pseudo- and actual codes of the solvers, and use debuggers and the traditional *printf* method, but they are insufficient to comprehend what is going on.

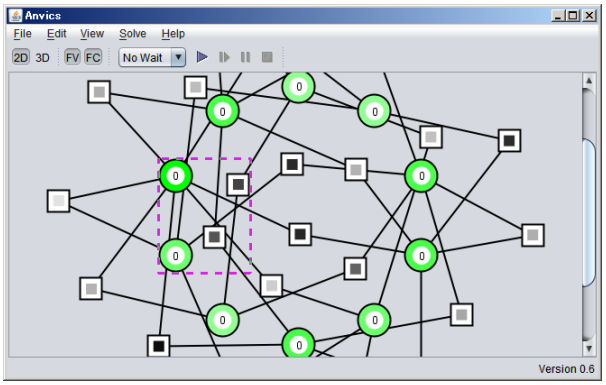

Fig. 1. The main window of Anvics. It contains a pane to show a constraint graph and a toolbar to control solvers.

We had investigated related work for improving the situation. Sadaoui et al. have proposed a tool [4], which enables us to generate and solve CSPs. JCLEditor is another graphical tool for developing and solving CSPs and soft CSPs including FCSPs [5]. However, neither of them provides functions for helping us understand the behavior of the solvers. They provide means to construct problems and obtain their solutions, but what happens in the solving processes is not transparent.

In this paper, we propose a tool named *Anvics* for analyzing and visualizing FCSPs (**Fig. 1**) [6]. This tool provides two- and three-dimensional views of constraint graphs (a representation of FCSPs). It enables users to comprehend intuitively the behavior of the solvers with sophisticated animation, and to debug in an environment similar to integrated development environments (IDEs) for programming languages. In addition, it provides a method for detecting the infinite-loop behavior of the solvers. We designed the tool for researchers, novice users, and practitioners of FCSPs.

In what follows, we first review the FCSPs, and present the details of the tool. Then we discuss the advantage of the tool, and lastly conclude with future work.

2. Fuzzy Constraint Satisfaction Problems

A CSP is a generic term for search problems to find combinations of values which satisfy the given constraints. It consists of the following components: a set of *variables* $X = \{x_1, \ldots, x_n\}$, a set of finite *domains* $D = \{D_1, \ldots, D_n\}$, and a set of *constraints* $C = \{c_1, \ldots, c_r\}$. Each variable x_i is supposed to take a value from the domain D_i. A constraint c_k denotes a relation R_k on a subset S_k of X, i.e.,

$$R_k \subseteq D_{k_1} \times \cdots \times D_{k_w} \text{ for } S_k = \{x_{k_1}, \ldots, x_{k_w}\}. \quad . \quad (1)$$

It represents permissible combinations of values to each variable of S_k. S_k is called the *scope* of R_k. If $w = 1$, 2, or 3, the constraint is called a unary, binary, or ternary constraint respectively. An *assignment* v to a scope S_k is denoted by $v[S_k] \in D_{k_1} \times \cdots \times D_{k_w}$. If $v[S_k] \in R_k$, then v *satisfies* the constraint c_k; otherwise, it *violates* c_k. To find a solution to a CSP involves the search for an assignment $v[X]$ that satisfies all the constraints of C.

An FCSP introduces a soft constraint called a *fuzzy constraint*, which need not be fully satisfied, but instead, its *degree of satisfaction* (satisfaction degree) should be considered as follows. In an FCSP, a constraint c_k denotes a fuzzy relation with its membership function defined by

$$\mu R_k : \prod_{x_i \in S_k} D_i \to [0,1]. \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad (2)$$

In other words, the membership value is defined by an assignment $v[S_k]$ to the scope S_k. This value is called the *satisfaction degree* of the fuzzy constraint.

Since an FCSP requires the satisfaction of the fuzzy conjunction of all fuzzy constraints, the satisfaction degree of the whole FCSP is defined as the smallest satisfaction degree of all the constraints as follows:

$$C_{\min}(v) = \min_{1 \le k \le r} \Big(\mu R_k \big(v[S_k] \big) \Big). \quad . \quad . \quad . \quad . \quad . \quad . \quad . \quad (3)$$

If $C_{\min}(v) > 0$, then v is called a *solution* of the FCSP, and a solution that maximizes $C_{\min}(v)$ is called an *optimal* solution. Therefore, an FCSP is regarded as an optimization problem that requires finding the assignment that maximizes the smallest satisfaction degree of the constraints.

The structure of an FCSP can be represented by a constraint graph, a dual constraint graph, and a hybrid constraint graph respectively. In a *constraint graph*, nodes and edges correspond to variables and constraints (**Fig. 2-a**). If a constraint c_k is binary, the two nodes in its scope are connected by an edge. If c_k is unary, the node in its scope is connected to itself by an edge as a self-loop. If c_k is ternary or of higher order, c_k is represented by a hyperedge, and the graph becomes a hypergraph. For such cases, it is convenient to use a *dual constraint graph*, where nodes and edges represent constraints and the nonempty intersections of their scopes respectively (**Fig. 2-b**). On the other hand, in a *hybrid constraint graph*, round nodes and square nodes represent variables and constraints respectively, and each edge connects a constraint node to the nodes of variables in its scope (**Fig. 2-c**). The hybrid constraint graphs have two features: the first is that with ordinary (i.e., non-hyper-) edges, they can show structures of FCSPs containing constraints that are ternary or of higher order and facilitate handling of FCSPs. The second is that the nodes can visually indicate the alteration of the states (properties) of the variables and constraints when their appearance is appropriately designed (as mentioned in Section 3.1).

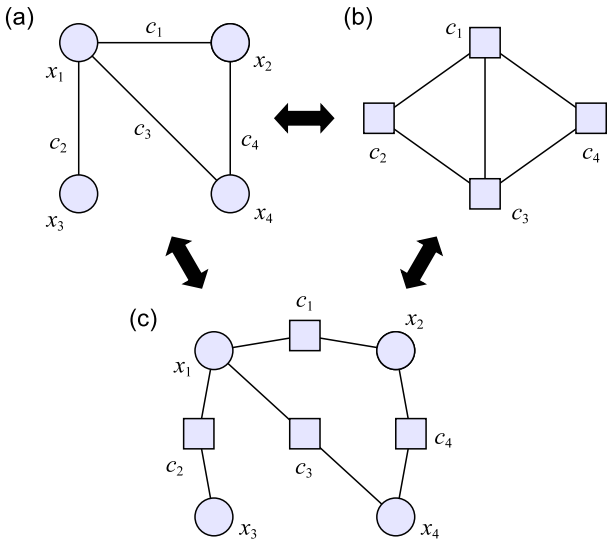

Fig. 2. A constraint graph (a), a dual constraint graph (b), and a hybrid constraint graph (c) of the same problem. In each graph, round nodes and square nodes represent variables and constraints respectively.

3. A Tool for FCSPs

Anvics, the tool we present in this paper (**Fig. 1**), can visualize with animation the solvers' behavior such as which variables they select, what values they assign, and how they change the satisfaction degrees of the constraints. We implemented the tool in Java with our FCSP library Stlics [7], which we had developed for our previous work [2, 3]. The tool allows users to load almost any problems generated by the Stlics library and run the solvers to obtain the solutions for the problems.

Anvics provides the users with ten solvers implemented in the Stlics library. For solving crisp problems, users can use the following five solvers: *breakout* [8], the crisp SRS 3,[1] *forward checking* [9], GENET [10], and *local changes* [11]. For solving fuzzy problems, users can use the following five solvers: *flexible local changes* [12], the fuzzy breakout,[1] the fuzzy forward checking,[1] fuzzy GENET [13], and SRS 3 [14].

1. In the list, some solvers without any references are converted versions we developed in order to adapt crisp solvers and fuzzy solvers to fuzzy and crisp problems respectively.

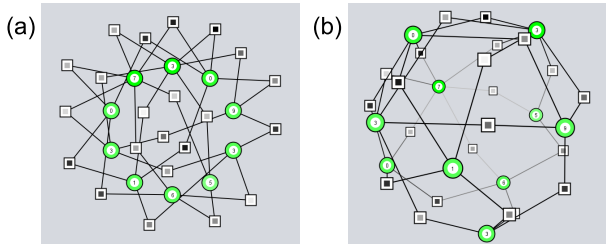

Fig. 3. (a) The two-dimensional (2D) view and (b) three-dimensional (3D) view of a hybrid constraint graph that represents the same FCSP, where round nodes and square nodes represent variables and constraints respectively.

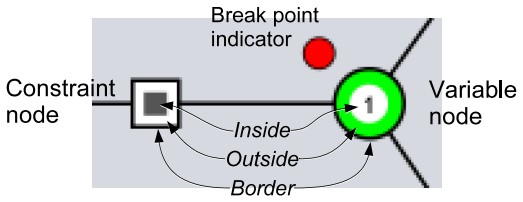

Fig. 4. A variable node and a constraint node. The area of each node is separated into the three parts and represents different properties in different colors. The red circle shows that a break point is assigned to the variable node.

3.1. Visualization Functions

Anvics depicts hybrid constraint graphs in two types of views: the two-dimensional (2D) view and the three-dimensional (3D) view (**Fig. 3**). In the 3D view, the tool literally shows graphs in a 3D space where nodes are placed on the surface of a virtual sphere. Since nodes and edges are less overlapping in this view than in the 2D view, it helps users understand the rough structures of the problems. In both views, as mentioned in Section 2, round nodes and square nodes represent variables and constraints respectively. Graphs in the views can be scaled by mouse wheel rotation, and be scrolled in the 2D view or rotated in the 3D view, by drag operation.

The layout of the nodes is adjusted by users' drag-and-drop operations, two alignment commands (circle alignment and tree alignment), and an automatic layout method. The circle alignment command puts variable nodes on a specific circle equally spaced from their adjacent nodes. On the other hand, the tree alignment command places variable nodes on the node positions of a tree structure where a variable node is regarded as the root and every pair of *neighbor* variable nodes in the original graph is mapped to a pair of a parent node and a child node in the tree. The two commands move only variable nodes.

In order to lay out all nodes in graphs automatically and to place constraint nodes properly after applying the alignment commands, we adopted the *modified spring-embedder model* [15] and its extension for the 3D view. The model associates each pair of graph nodes with a virtual spring. The equations for calculating the forces of springs are as follows:

$$f_a(d) = c_a \frac{d^2}{k},\ f_r(d) = -c_r \frac{k^2}{d}, \quad \ldots\ldots\ (4)$$

where f_a and f_r are attractive and repulsive forces among nodes, d and k are the actual and the ideal distances between two nodes, and c_a and c_r are constants that decide the strength of those forces respectively. The attractive force f_a takes effect between a variable node and a constraint node directly connected with each other, and the repulsive force f_r does between those unconnected. The process of laying out is animated, and users can watch the process to control the constants of the forces and the radius of the sphere interactively.

The area of the nodes is separated into three visual parts: an inside, an outside, and a border, which are painted with different coloring schemes for visualizing specific properties of variables and constraints (**Fig. 4**). The properties are categorized into static ones and dynamic ones. The static properties are the valence (the number of related constraints) of a variable and the order (the size of the scope) of a constraint, which will not change once the problems have been constructed. The dynamic properties are the assignment count (the number of assignments done in each variable) and the domain size of a variable; and the satisfaction degree of a constraint, which will change while a solver is running. Through a dialog box, users can associate each visual part of the nodes with a specific property they want to observe. If one of the dynamic properties is associated with a visual part, the color of the part will change in response to the alteration of the properties while solving processes.

Users can observe the assignment of values to variables by the animation with flashing nodes. In graphs, when a solver assigns a value to a variable, the corresponding variable node flashes in yellow, or its color changes to yellow instantaneously and then gradually turns back to the original color. Likewise, a constraint node also flashes when values are assigned to variables of its scope.

3.2. Analysis Functions

For checking the properties of FCSPs at any time, Anvics provides users with the inspector dialog (**Fig. 5**). The inspector shows the properties of selected variables and constraints, and the entire problem in the form of lists. It updates the display of the properties soon after a solver changes one of them, for supporting users to know the current state of the problem in the solving process.

As a means of controlling solvers, the tool provides the ways of setting termination conditions, the user-interactive solver controller, and break points. Before running a solver, users can specify termination conditions: the target satisfaction degree, the maximum step counts, and the limit time, through a dialog box (**Fig. 6**). This function is used for running solvers that do not stop without specifying those conditions, even though users can stop them manually with the solver controller.

By operating the solver controller and putting break points on variables, users can intervene in the execution of solvers (**Fig. 7**). When a solver is running, by using

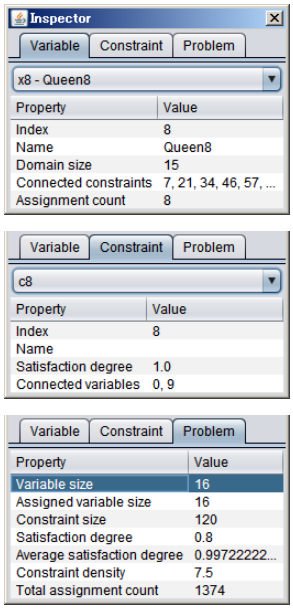

Fig. 5. The inspector dialog box. Users can see the properties of variables, constraints, and entire problems respectively on each tab. After a solver changes one of the properties, the inspector updates its display immediately.

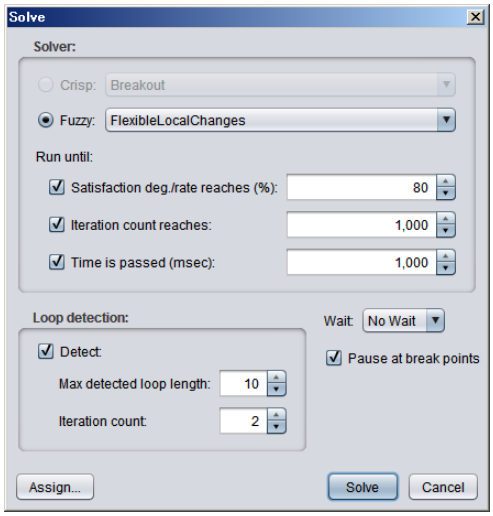

Fig. 6. The solve dialog box. Through the dialog box, users can select solvers, and set termination conditions, the loop detector, and a wait time.

the controller on the toolbar or the menu of the main window, users can have the solver stop, pause, and run step by step (every one step, five steps, or ten steps). To observe the properties of nodes that are frequently updated by the solver and the variable that the solver selects for assignment at each step, users can slow down the effective running speed of the solver by setting a wait time. The solver will pause for this specified time whenever it assigns a value to a variable. In addition, if users need to be notified when a value is assigned to a specific variable, they can put a break point on the variable node by clicking a mouse wheel. Hereby, a solver will always pause just after it has assigned a value to the specified variable (**Fig. 4**). After the solver pauses, users can resume the execution of the solver step by step from that time.

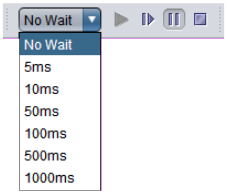

Fig. 7. The controller for solver execution. Like an audio playback control, users can control solvers clicking these buttons on the toolbar of the tool.

Anvics provides a detector for infinite loops of assignments, which means that a solver iterates the same sequence of assignment actions infinitely, and they can be seen mainly when running stochastic solvers. When users want to activate the detector, they need to decide its parameters, the maximum loop length (MLL) and the iterating count (IC), through the solver dialog (**Fig. 6**). The MLL restricts the length of the loop to be detected; greater MLL value means greater detection power at the greater search cost. The IC is used to regard a finite number ($=$ IC) of repetitions as an infinite loop. When a loop is detected (a solver repeats IC times the same assignment sequence with its length not exceeding MLL), a dialog box pops up to notify this information and the solver turns into the pause mode. Although the loop detector could be implemented in solvers directly, it may reduce their efficiency not only in analysis but also in practical use. Instead, Anvics allows users to detect such loops only when the users are analyzing problems, even if solvers where no detectors are implemented cause such loops.

3.3. Examples of Applying Solvers

In this section, we show how Anvics visualizes the behavior of solvers by using two solvers (forward checking and breakout) for solving a sample problem, eight queens on an 8×7 board[2] (**Fig. 8**). The problem is simple and often used as an example of an FCSP, and thus, the solvers can show the differences of the two solvers explicitly.

Forward checking (FC) is a representative of the systematic solvers. In the experiment, we arranged Anvics so that it would be changing the colors of the variable nodes according to their domain size and those of the constraint nodes according to their satisfaction degree (**Fig. 8-a**). In the beginning, all constraint nodes were painted in red, showing that the satisfaction degrees were undefined because the solver had cleared all the variables. By variable node colors thinning gradually, users were able to observe how the solver was pruning the domains while backtracking. In addition, the users observed that the nodes were

2. The *eight queens on an* 8×7 board is a fuzzy extension of the eight-queen puzzle. Like the original puzzle, the queens *should* be placed so that no queen attacks any other, and if that is impossible, each pair of the queens should be placed at points distant from each other.

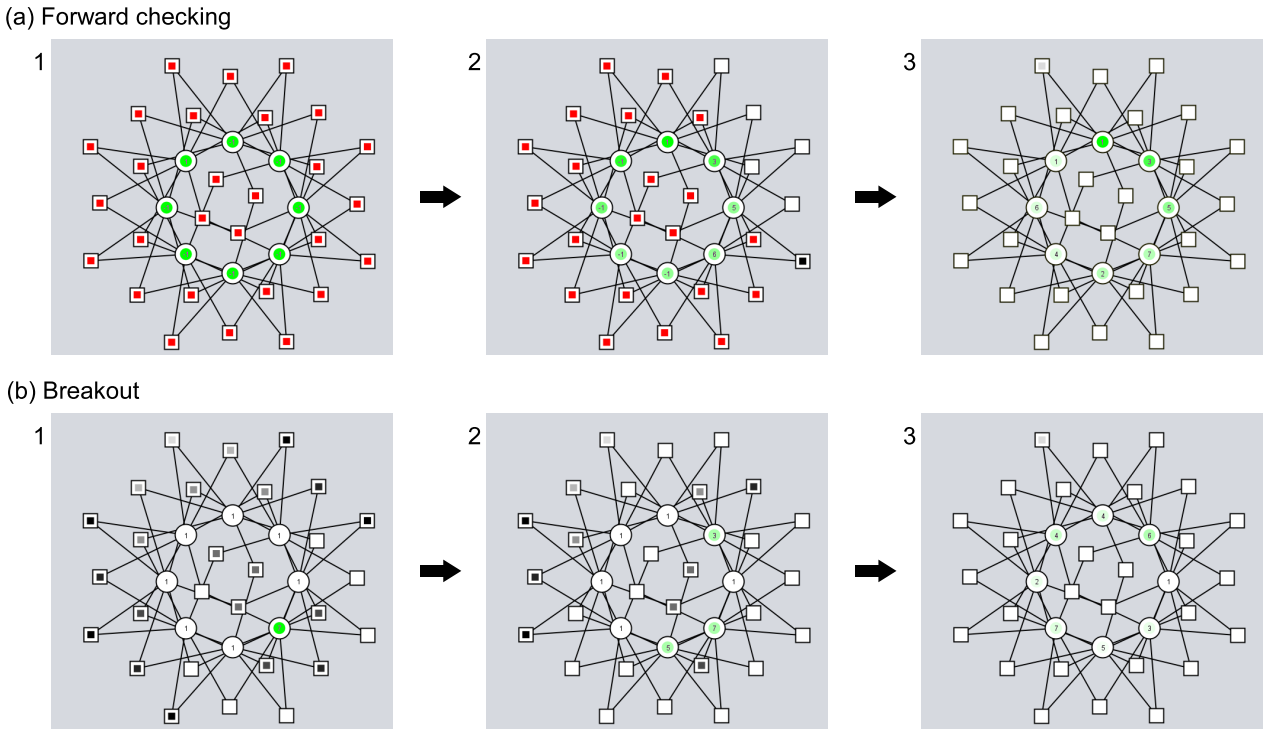

Fig. 8. Hybrid constraint graphs of 8 queens on an 8×7 board problem during applying (a) forward checking and (b) breakout. In each row, the leftmost graphs are in the states after a few steps after the solvers started, the rightmost ones are in the states after the solvers finished, and the middle ones are in the states between the lefts and the rights.

flashing in clockwise and anticlockwise from the topmost position with regularity (because the solver was performing systematically), and they were able to notice that the solver was searching in a depth-first way.

Breakout is a representative of the stochastic solvers. In the experiment, we arranged Anvics so that it would be changing the colors of the variable nodes according to their assignment count (**Fig. 8-b**). Compared to the experiment with FC, at the beginning, the constraint nodes were painted in gray, meaning that the satisfaction degrees were defined because the initial values had been assigned to the variables. At first, the users mistakenly observed that the nodes were flashing completely at random (because the solver was performing stochastically). However, after an additional short period of observation on changing colors of the constraint nodes, they finally understood the fact that the solver was regularly selecting the variables related to the worst constraints.

4. Conclusion

We have proposed a tool, Anvics, which allows FCSP users to grasp the structures of the problems and the behaviors of the solvers intuitively. One of the most notable features of the tool is the visualization facility to demonstrate with animation how the solvers behave in the course of the tree search and the local search computation. Other features are the 3D view of constraint graphs and the function of infinite-loop detection. In addition, the tool enables users to debug FCSP models in an environment similar to integrated development environments (IDEs) for commonly-used programming languages.

We hope that the visualization realized in this paper not only offers debugging facilities but also leads to discoveries of new knowledge of the behavior of FCSP solvers. For that purpose, we are planning to add functions to visualize other statistical information, such as the number of times when the satisfaction degree of a constraint becomes the worst in the problem and the availability of values in domains in a solving process. We anticipate that visualizing such information will bring to users deeper knowledge of the solvers. In addition, since the design of domains and membership functions of FCSPs governs the behavior of the solvers, we are also planning to implement adjustment functions of such elements of FCPSs.

References:

[1] Z. Ruttkay, "Fuzzy Constraint Satisfaction," In Proc. of the 3rd IEEE Conf. on Fuzzy Systems, Vol.2, pp. 1263-1268, Orlando, FL, USA, 1994.

[2] T. Yanagida and H. Nonaka, "Flexible Widget Layout Formulated as Fuzzy Constraint Satisfaction Problem," In K. Nakamatsu, G. Phillips-Wren, L. C. Jain, and R. J. Howlett, editors, Proc. of the 1st KES Int. Symposium on Intelligent Decision Technologies (KES IDT 2009), New Advances in Intelligent Decision Technologies, pp. 73-83, Himeji, Japan, Springer, April 2009.

[3] T. Yanagida, Y. Sudo, and H. Nonaka, "Flexible Widget Layout Based on Fuzzy Constraint Satisfaction," J. of Japan Society for Fuzzy Theory and Intelligent Informatics, Vol.20, No.6, pp. 840-849, December 2008. (in Japanese)

[4] S. Sadaoui, M. Mouhoub, and X. Li, "An OCL-Based CSP Specification and Solving Tool," In N. T. Nguyen and R. Katarzyniak, editors, New Challenges in Applied Intelligence Technologies, Vol.134

of Studies in Computational Intelligence, pp. 235-244, Springer, 2008.
[5] L. Grangier. "JCLEditor – A Graphical CSP Solver Based on JCL," on the Web, 2005. Available at http://liawww.epfl.ch/JCL/
[6] T. Yanagida. "Download Site for Anvics," 2009. Available at http://kussharo.complex.eng.hokudai.ac.jp/~takty/demo/anvics.en.html
[7] T. Yanagida and Y. Sudo. "Stlics: A Java Library for Fuzzy Constraint Satisfaction Problems," 2009. Available at http://kussharo.complex.eng.hokudai.ac.jp/~takty/interest/stlics.en.html
[8] P. Morris, "The Breakout Method for Escaping From Local Minima," In Proc. of the 11th National Conf. on Artificial Intelligence (AAAI-93), pp. 40-45, Washington, DC, USA, AAAI Press/The MIT Press, 1993.
[9] R. M. Haralick and G. L. Elliott, "Increasing Tree Search Efficiency for Constraint Satisfaction Problems," Artificial Intelligence, Vol.14, No.3, pp. 263-313, 1980.
[10] A. J. Davenport, E. P. K. Tsang, C. J. Wang, and K. Zhu, "GENET: A Connectionist Architecture for Solving Constraint Satisfaction Problems by Iterative Improvement," In Proc. of the 12th National Conf. on Artificial Intelligence (AAAI '94), pp. 325-330, 1994.
[11] G. Verfaillie and T. Schiex, "Solution Reuse in Dynamic Constraint Satisfaction Problems," In Proc. of the 12th National Conf. on Artificial Intelligence, pp. 307-312, Seattle, WA, USA, AAAI Press, 1994.
[12] I. Miguel and Q. Shen, "Extending FCSP to Support Dynamically Changing Problems," In Proc. of the 8th IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE '99), Vol.3, pp. 1615-1620, Seoul, Korea, 1999.
[13] J. H. Y. Wong and H. Leung, "Extending GENET to Solve Fuzzy Constraint Satisfaction Problems," the 15th National Conf. on Artificial Intelligence/10th Conf. on Innovative Applications of Artificial Intelligence, (AAAI '98/IAAI '98), pp. 380-385, Menlo Park, CA, USA, 1998.
[14] Y. Sudo and M. Kurihara, "Spread-Repair-Shrink: A Hybrid Algorithm for Solving Fuzzy Constraint Satisfaction Problems," In Proc. of the 2006 IEEE World Congress on Computational Intelligence/the 2006 IEEE Int. Conf. on Fuzzy Systems (WCCI 2006/FUZZ-IEEE 2006), pp. 2127-2133, Vancouver, BC, Canada, 2006.
[15] T. M. J. Fruchterman and E. M. Reingold, "Graph Drawing by Force-Directed Placement," Software – Practice & Experience, Vol.21, No.11, pp. 1129-1164, Nov. 1991.

Name:
Takuto Yanagida

Affiliation:
Postdoctoral Fellow (supported by the Global COE Program), Graduate School of Information Science and Technology, Hokkaido University

Address:
Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan

Brief Biographical History:
2009 Ph.D. at Hokkaido University
2009- Post Doctoral Fellow, Formative System Engineering Laboratory, Hokkaido University
2009.12- Assistant Professor, Shizuoka University

Main Works:
• "Flexible Widget Layout Based on Fuzzy Constraint Satisfaction," J. of Japan Society for Fuzzy Theory and Intelligent Informatics, Vol.20, pp. 840-849, 2008.

Membership in Academic Societies:
• The Information Processing Society of Japan (IPSJ)
• The Institute of Electronics, Information and Communication Engineers (IEICE)
• The Association for Computing Machinery (ACM)

Name:
Masahito Kurihara

Affiliation:
Professor, Graduate School of Information Science and Technology, Hokkaido University

Address:
Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan

Brief Biographical History:
1980- Joined Hokkaido University as Instructor
1989- Associate Professor, Hokkaido University
1996- Professor, Hokkaido Institute of Technology
2002- Professor, Hokkaido University

Main Works:
• "Completion for Multiple Reduction Orderings," J. of Automated Reasoning, Vol.23, pp. 25-42, 1999.

Membership in Academic Societies:
• The Information Processing Society of Japan (IPSJ)
• The Institute of Electronics, Information and Communication Engineers (IEICE)
• The Japanese Society for Artificial Intelligence (JSAI)
• American Association for Artificial Intelligence (AAAI)

Name:
Hidetoshi Nonaka

Affiliation:
Associate Professor, Graduate School of Information Science and Technology, Hokkaido University

Address:
Kita 14, Nishi 9, Kita-ku, Sapporo 060-0814, Japan

Brief Biographical History:
1993 Ph.D. at Hokkaido University
1985- Instructor at Hokkaido University
1996- Associate Professor, Hokkaido University

Main Works:
• "Operation-Based Notation for Archimedean Graph," J. of Systemics, Cybernetics and Informatics, Vol.6, No.6, pp. 19-22, 2008.

Membership in Academic Societies:
• The Society of Instrument and Control Engineers (SICE)
• The Information Processing Society of Japan (IPSJ)
• The Institute of Electronics, Information and Communication Engineers (IEICE)
• Japan Society for Fuzzy Theory and Intelligent Informatics (SOFT)
• Human Interface Society (HIS)
• Japan Ergonomics Society (JES)