International Journal of Innovative Computing, Information and Control
Volume **11**, Number **1**, February **2015**

ICIC International ©2015 ISSN 1349-4198
pp. 137–151

EFFICIENT SOLUTION OF CONSTRAINT SATISFACTION PROBLEMS BY TAKING INTO ACCOUNT THE RELATIONSHIP OF CONSTRAINTS

Hiroshi Mabuchi[1] and Kohei Fukuchi[2]

[1]Faculty of Software and Information Science
Iwate Prefectural University
152-52 Sugo, Takizawa, Iwate 020-0693, Japan
mabu@iwate-pu.ac.jp

[2]ECHNA Corporation
1-6-30, Chuodori, Morioka, Iwate 020-0021, Japan
kfukuchi@echna.co.jp

Received March 2014; revised July 2014

Abstract. *This paper proposes an efficient method to solve number-place problems. Finding efficient solutions for these problems is very important as from this work, efficient solutions of other constraint satisfaction problems may be derived. Concretely, this paper proposes a comprehensive method by taking into account the relationship between constraints existing in any of the rows, columns and sub-grids. This study adopts the equivalent transformation (ET) model which is a framework to solve problems, where without depending only on a constraint-solving algorithm possessed in a system, users can add new efficient rules and data structures for constraint-solving and consequently the constraint-solving algorithm is improved. Also, in order to demonstrate the effectiveness of the proposed method, we compare its computation result against that of other methods.*

Keywords: Constraint satisfaction problem, Equivalent transformation, Transformation rule, Constraint processing, Global processing

1. **Introduction.** Recently, numerous studies have been conducted on solving constraint satisfaction problems (CSPs) [4, 6, 12, 13, 14, 16, 17]. In particular, a method to solve number-place problems is extensively studied worldwide. Finding efficient solutions for these problems is very important as from this work, efficient solutions of other constraint satisfaction problems may be derived.

Constraint atoms and algorithms are used by the system to solve problems in constraint logic programming (CLP) [3, 5, 8, 17]. Using the approach of combining constraint atoms enables users to program and offers the advantage of guaranteeing the solving method's correctness because the system provides a constraint-solving algorithm.

However, when working in CLP, expressive power for programming is limited due to its dependency on a pre-supported constraint-solving algorithm. If limits are violated, there may be a drastic decrease in computation efficiency. When this happens, the approach is unable to improve constraint-solving algorithms.

In Constraint Handling Rules (CHR) [7] which is a kind of expansion of CLP, users can define some rules to solve constraints, the correctness of which is assured based on logical inference. However, making use of user-defined rules to solve constraints presents difficulties in assuring to the correctness of solving method.

This paper adopts the equivalent transformation (ET) model [1, 2, 10, 19, 20] which is a framework to solve problems, where without depending only on a constraint-solving

algorithm possessed in a system, users can add new efficient rules and data structures for constraint-solving and consequently the constraint-solving algorithm is improved. In the ET model, the correctness of rules can be assured without considering interrelations with other rules, therefore, as long as correct ET rules are added to a program, the correctness of the entire program (correctness of computation results) can be assured without re-examining the entire program. The framework mentioned above, as it was applied in this study, not only enables the easy creation or modification of new rules, but also assures the correctness of computation results.

Based on this framework, the conventional method [9, 11, 15, 17] by which problem-solving rules for number-place problems are applied to the smallest unit was modified in the previous study [18]. Specifically, the previous study did not apply rules to the smallest unit. Instead, it utilized global processing for each row, column and sub-grid to improve the computation efficiency.

However, in number-place problems, even with the effective rules developed in the previous study, solving problems becomes more difficult and the required computation time may significantly increase as problems become more complicated. This is because the method used in the previous study was one of successively simplifying the problems in each row, column and sub-grid without taking into account the relationships between constraints existing in any of the rows, columns and sub-grids. Therefore, when simplification of problems becomes challenging, rules must be applied to split the problem state, thus greatly increasing the amount of computation. In order to overcome this issue while also making computations more efficient, it is essential that rules which take into account the relationship among any of the rows, columns and sub-grids are created as the primary rules.

In this study, we propose a comprehensive method, an approach different from the previous study [18], by taking into account the relationship between constraints existing in any of the rows, columns and sub-grids. Then, we will demonstrate the effectiveness of the proposed method through number-place problem experiments. In particular, we studied relationships among the rows, columns and sub-grids and developed a method which makes use of this relationship to simplify problems. Then, we created new rules relevant to this relationship. These studies and preferred-basis usage of the rules relevant to the relationship among the rows, columns and sub-grids solves the issue observed in the previous study while making computation more efficient. Also, in order to demonstrate the effectiveness of the proposed method, we compare its computation result against that of logic programming languages, referred to as Prolog [9, 15], and the previous study's method.

2. **Computation by Equivalent Transformation.**

This study adopts the equivalent transformation (ET) framework for problem solving, where computation is regarded as equivalent transformation of declarative descriptions [1].

2.1. **Problem solving by equivalent transformation.**

A declarative description is successively simplified into different declarative descriptions by ET rules, and from a finally simplified declarative description a solution may be obtained [1]. If correct ET rules are used in all transformation steps, an answer is guaranteed to be correct.

2.2. **Characteristics of ET computation model.**

ET computation model has the following characteristics.

This model can use not only unfolding rules [15] but also other various Et rules as transformation rules. By nondeterministic selection of an ET rule at each step of computation, various computations become possible.

The correctness of computation is assured in problem solving by equivalent transformation. And the correctness of a rule can be assured without considering interrelations with other rules, therefore, as long as correct ET rules are added to a program, the correctness of the entire program (correctness of computation results) can be assured without re-examining the entire program.

In addition, ET computation model has the following advantages.

- It desclibes various expressive rules since it offers abundant data structures.
- It controlls processing flexibly since the order of computation is not fixed.
- It improves algorithms at a lower cost by the addition and deletion of ET rules.

3. Formalization of Constraint Satisfaction Problem.

3.1. **Example of constraint satisfaction problem.** In this study, Number-place problems are used as examples of constraint satisfaction problems. Number-place problem is a puzzle in which numbers from 1 to 9 are placed in each small blank square (Figure 1). In this study, the entire number-place problem table is called a "grid". It is made up of nine "sub-grids," each with a thicker border, in a 3×3 pattern. Each sub-grid is made up of nine "boxes" in a 3×3 pattern. A box is defined as the field in which the numbers 1 to 9 are placed.

There are two constraints:

(Constraint 1): The numbers 1 through 9 will be placed into each small blank square.

(Constraint 2): The same number cannot be placed in any one column or row, nor within any one sub-grid surrounded by a thicker border.

8				6				
		1						3
	7	9				8		
3			2				7	
				7			5	
1			8				9	
	3	2				1		
			6					4
5				8				

FIGURE 1. Number-place problem

3.2. **Formalization of problems.** This section formalizes problems by declarative descriptions. Constraints to be satisfied when solving a number-place problem are to satisfy a given assignment of the problem (given_assignment predicate) and to adhere to the constraints of the problem (NP_constraints predicate). These are represented with the following clause. Symbols starting with "*" represent variables.

```
(answer *numberplace) ←
     (given_assignment *numberplace),
     (NP_constraints *numberplace).
```

"given_assignment" predicate is defined as follows. The "?" marks represent anonymous variables, each of which is different from all others.

```
(given_assignment *numberplace) ←
     (= *numberplace ( (8 ? ? ? 6 ? ? ? ?)
```

```
(? ? ? 1 ? ? ? ? 3)
(? 7 9 ? ? ? 8 ? ?)
(3 ? ? 2 ? ? ? 7 ?)
(? ? ? ? 7 ? ? 5 ?)
(1 ? ? 8 ? ? ? 9 ?)
(? 3 2 ? ? ? 1 ? ?)
(? ? ? 6 ? ? ? ? 4)
(5 ? ? ? 8 ? ? ? ?))).
```

A variable, `*numberplace`, is equal to the list which represents the given assignment of a problem. "NP_constraints" predicate is expressed in accordance with (Constraint 1) and (Constraint 2) in Section 3.1.

4. Outline of Prior Research.

This section describes the outline of prior research [18].

4.1. Solution.

In this section, we describe the solution in the prior research.

As shown in Figure 2, an initial problem is simplified so that there are individual sub-problems for each row, column and sub-grid. At this point, the size of each sub-problem is now smaller than that of the initial problem. Then, each sub-problem is simplified further, through application of certain rules, to obtain new problems. By repeating this process, the answer will be obtained.

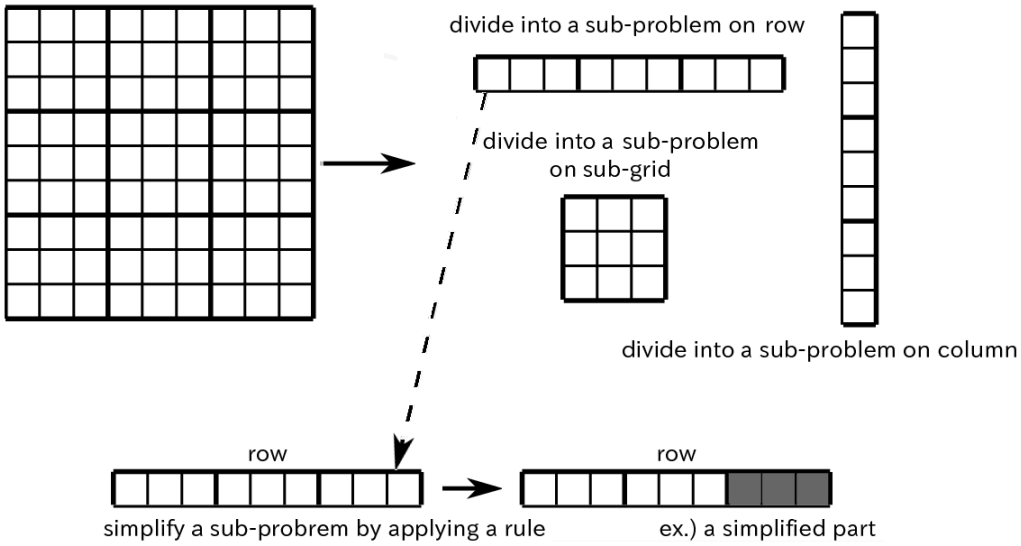

FIGURE 2. Solution method in prior research

4.2. Rules.

AllDifferent atom representing "Constraint 2" (see Section 3.1), providing that elements of the list are different from each other.

- **"candidate elimination" rule.** Elements that are identical to the numbers are all eliminated from the candidates of other i-vars.

```
(AllDifferent (3 *a~(2 3 4) 5 *b~(2 3 4 5)))
                        ↓
(AllDifferent (*a~(2 4) *b~(2 4)))
```

An i-var is defined as a variable which has been given information. An i-var has the form in which a variable is followed by a symbol, " ," and ends with S-expressions such as "(1 2 3)," as with `*x~(1 2 3)`.

- **"unification" rule.** When an i-var reduces candidates to one number, the variable is unified with the number.

```
(AllDifferent (*a~(2 3 4) *b~(2 3 4) *c~(3)))
                    ↓
(AllDifferent (*a~(2 3 4) *b~(2 3 4) 3))
```

- **"number decision" rule.**

```
(AllDifferent *a~(5 7 9) *b~(5 9) *c~(4 5 9) *d~(4 5))
```

  In this example, variable `*a` can be determined as 7.

4.3. **Results and problems.** An AllDifferent atom is not transformed into other smaller (simpler) atom, but instead an AllDifferent atom is regarded as the smallest unit and tries to do candidate elimination. Since information in AllDifferent atom is more than that in other small atom, candidate elimination can be efficiently performed by good use of the information. In the prior study, the use of AllDifferent atoms generated better results in terms of processing time and number of rule applications [18].

In number-place problems, even with the effective rules described in the previous section, solving problems becomes more difficult and the required processing time may significantly increase as problems become more complicated. This is because the method used in the prior study [18] was one of successively simplifying the problems in each row, column and sub-grid without taking into account the relationship among any of the rows, columns and sub-grids. That is, the relationship between any of the rows and columns, between any of the rows and sub-grids or between any of the columns and sub-grids was not taken into account. Therefore, when simplification of problems becomes challenging, rules must be applied to split the problem state, thus greatly increasing the amount of computation.

In number-place problems, take the case of two rows being selected. When studying the relationship between these two rows, it is important to take into account the information regarding the columns and sub-grids relevant to those rows and examine the interrelations of each row, column and sub-grid. This approach makes the computation more efficient and comprehensive than the prior approach [18] where the problems in each row, column and sub-grid were successively simplified and numbers that did not contradict the constraints were determined during the process.

5. Proposed Method.

5.1. **Significance in framework.** In the ET programming framework, the addition, elimination or modification of ET rules has important implications with regard to low-cost system improvement while still preserving correctness, an effect which cannot be derived in other frameworks. This is due to the following reasons.

For example, in the logic programming framework, even when ET rules are added and incorporated into the system, correct computation cannot be readily performed. This is because for computation procedures (e.g., SLD resolution) operating under the logic programming framework, computation cannot be performed unless the correctness of the entire resolution, which is a collection of individual inference rules, is guaranteed. That is, it is not sufficient to simply add individual rules that have correctness but instead the correctness of the entire resolution needs to be re-proved each time a rule is added [9, 15]. This applies when ET rules are eliminated or modified.

On the other hand, in the framework of ET programming, the correctness of a rule can be assured without considering interrelations with other rules, therefore, as long as correct ET rules are added to a program, the correctness of the entire program (correctness of computation results) can be assured without re-examining the entire program. Because

of this, ET rules can be easily added, eliminated or modified and can therefore provide low-cost improvements to a system (program) while still preserving its correctness.

Also, by designating an order of priority to each rule, problems are simplified through the application of rules that have higher priority. Therefore, by compiling, without deletion, rules other than those that are unnecessary and by handling the program as rules' database, it is possible to reuse rules, to perform highly flexible computations and to easily construct a system that can handle a wide variety of problems.

5.2. **Summary of the proposed method.** In this section, a new method is proposed to solve the problems described in Section 4.3.

As shown in Figure 3, it is a comprehensive method, an approach different from the prior study [18], that takes into account any relationships among the rows, columns and sub-grids. At this time, the numbers from 1 to 9 are further simplified into number-specific problems through a priority basis application of the rules that take into account any relationships among the rows, columns and sub-grids. Because the problem is simplified into sub-problems, each of which is specific to a specific number from 1 to 9, the total size of the simplified sub-problems is the same as that of the initial problem.

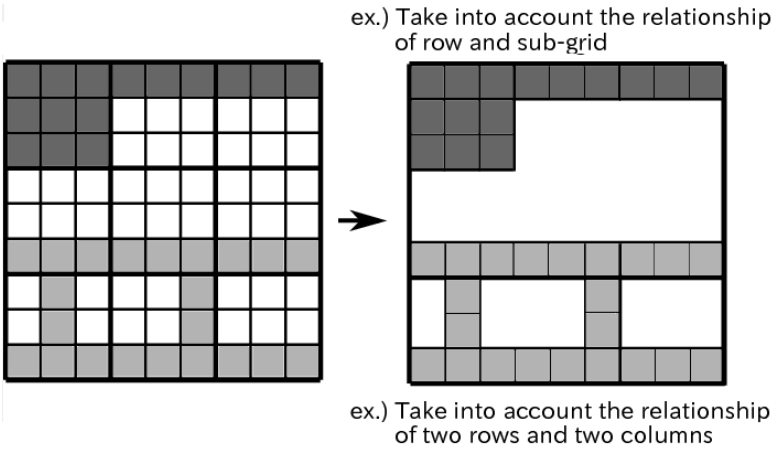

FIGURE 3. Proposed method

First, relationships among the rows, columns and sub-grids are studied, and a method which makes use of this relationship to simplify problems is developed. Then, new rules relevant to this relationship are created. These studies and preferred-basis usage of the rules relevant to the relationship among the rows, columns and sub-grids can solve the issue observed in the prior study while making computation more efficient [18].

5.3. **Concept of problem simplification.** The proposed method takes numbers and a set of boxes (variables) as individual units of the problems and simplifies the problems while maintaining any relationships among the rows, columns and sub-grids.

For example, using the example problem given in Figure 4, when determining at which box where the number 1 can be placed (1-question), it is clear that 1 cannot be placed in boxes on the rows, columns and sub-grids at which 1 is already placed. Therefore, the need to check whether a box can accept the number 1 is not required and the information of these boxes can be eliminated. It is also clear that any boxes which already have numbers cannot accept any other numbers; therefore, the information of these numbers can be eliminated as well. The same process is applied to other numbers (2 to 9) as well.

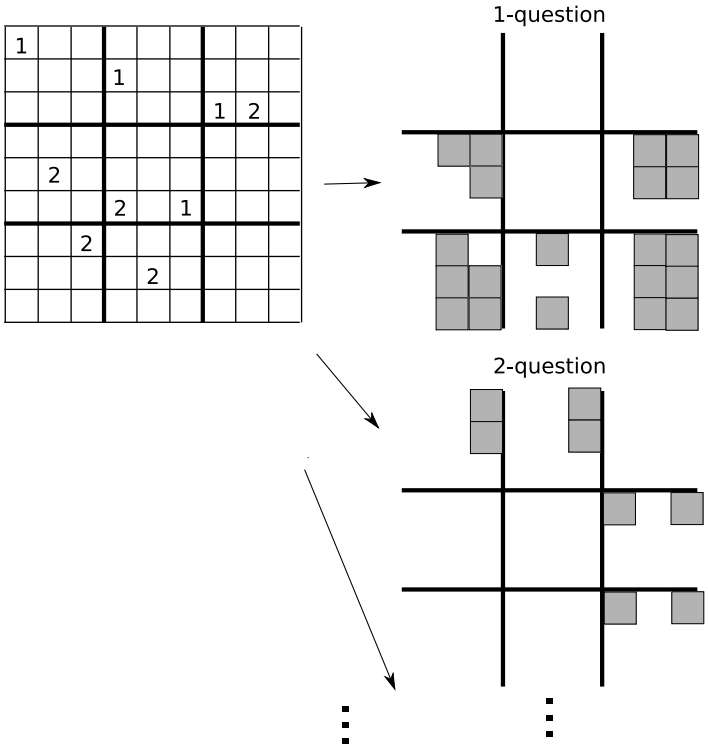

FIGURE 4. Concept of problem simplification

As the information regarding the rows, columns and sub-grids remains, rules relevant to their relationships can be applied to problems which have been simplified in this manner.

```
(given_question *numberplace) ←
     (Layer 1 *row-list *column-list *box-list)
     (Layer 2 *row-list *column-list *box-list)
                        ⋮
     (Layer 9 *row-list *column-list *box-list)
     (Layer All *row-list *column-list *box-list))).
```

A given_question atom contains one argument `*numberplace` which is applied in a number-place problem that has been formalized. The layer atoms have four arguments. The first argument assigns numbers (1 to 9) to each box and determines which box to refer to based on these numbers. Also, in the first argument, "All", which is anything other than the numbers 1 to 9, is added. This is because there may be cases where some boxes within the rows, columns and sub-grids must be referred to but this does not occur when a rule intended for multiple boxes is applied. To avoid this, a layer atom containing "All" in its first argument is used so that all boxes can be referred to. The second, third and fourth arguments contain lists of the simplified rows, simplified columns and simplified sub-grids respectively.

Instead of simplifying problems by row, column and sub-grid, the proposed method assigns numbers by the first argument (layer atom) and refers to boxes which contain the assigned numbers' information to simplify problems. Therefore, the relationship among any of the rows, columns and sub-grids can be maintained and the rules relevant to this relationship can be applied. This makes the computation more efficient because splitting of the problem state is kept at a minimum.

5.4. **Rules.** This section describes rules that were not used in the previous study [18] but are introduced for the first time under this method. Also, other rules (including those utilized in the previous study) which are required to solve the problem are explained in the Appendix (see Appendix).

- **"comprehension" rule**

  The "comprehension" rule is one which takes into account the relationship of the problem's row, column and sub-grid. Figure 5 is an example of how boxes to which the number 2 can be placed are checked. When checking by sub-grid, it can be noted that 2 can be placed either in H5 (at the row 5 in the column H) or H6. From this, when checking by column, 2 cannot be placed in H1, H2 or H3. Therefore, 2 can be eliminated from the number information of H1, H2 and H3 boxes (variables).

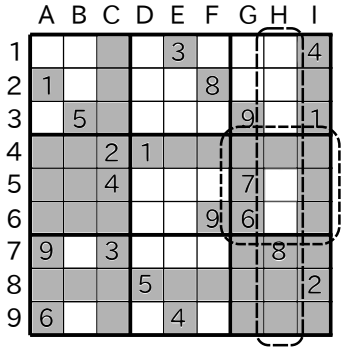

FIGURE 5. "comprehension" rule

In Figure 5, when examining column H, boxes to which 2 can be placed are {H1, H2, H3, H5, H6}. Also, when examining the sub-grid (one of the nine sub-grids with a thicker border) enclosed with dotted lines, the boxes that intersect column H are {H5, H6}. At this time, because the boxes that are within that sub-grid are also contained in column H, 2 can be eliminated from the number information attached to variables for the remaining boxes {H1, H2, H3}. An example of how to execute follows. Here, `*columnk`($1 \leq$ `k` ≤ 9) indicates each individual column and `*boxj`($1 \leq$ `j` ≤ 9) indicates each individual sub-grid.

```
(Layer 2 *row-list
  (*column1 *column2···
  (*a~(2 5 6 7) *b~(2 3 5 6 7) *c~(2 3 6 7)  *d~(1 2 3 5 9) *e~(1 2 3 4 5))
    ···  *column9)
  (*box1 *box2 ···
   (*d~(1 2 3 5 9) *e~(1 2 3 4 5))
   ··· *box9))
```

With regard to lists in the example above that contain the variables `*d` and `*e`, the lists for the columns carry out comprehension for the sub-grids' lists. At this time, the i-vars `*a` and `*b` as well as `*c` can be eliminated from the columns' lists. This rule is one which could not be used with the previous study's method which did not taken into account the relationships among the rows, columns and sub-grids.

- **"alliance" rule**

  The "alliance" rule is one which takes into account the relationship between any two boxes within any of the rows, columns and sub-grids (Figure 6). When two selected boxes (variables) both have two number candidates (A, B) and the candidates for those numbers are the same, their number information (A, B), which is attached

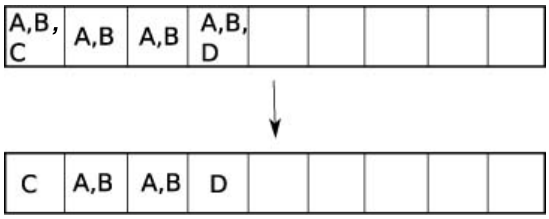

FIGURE 6. "alliance" rule

to those two boxes (variables), can be eliminated from the information of all boxes other than those two.

The "alliance" rule is one which refers to multiple i-vars and their attached number information. In reference to boxes to which number information assigned under the first argument as variables, this rule, which takes into account information from boxes (variables) that are not referred to, is likely to cause an error and therefore cannot be applied. Therefore, an accommodation to this is made by using a rule that assigns "All" in its first argument so that all boxes will be checked.

By its nature, the "alliance" rule must check i-vars and their attached number information as contained within lists and therefore has a large processing cost. Therefore, by lowering the priority of this rule's application, there are a lower number of uses of this rule which results in improved computation efficiency.

- **"occupancy" rule**

  The "occupancy" rule is one which takes into account the relationship between any two boxes within any of the rows, columns and sub-grids (Figure 7). This rule can only be applied when the numbers that are assigned in two selected boxes (variables) are different from the number candidates for other boxes (variables) and when there are only two numbers which appear only in those two selected boxes (variables). From the number information attached to those two selected boxes (variables), the numbers for those boxes (variables) can be determined.

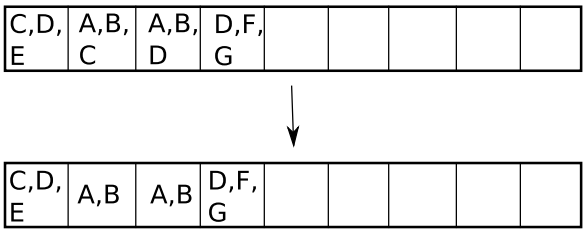

FIGURE 7. "occupancy" rule

The procedures for this rule are as follows. First, select any one of the rows, columns or sub-grids and select two boxes from the selected row, column or sub-grid. Next, pick the intersection for the numbers assigned to these boxes (variables) and make sure that the intersection does not appear within the numbers assigned to the selected row, column or sub-grid. Only on confirmation can the "occupancy" rule be applied and the information of those selected boxes (variables) be replaced with the numbers from the intersection.

Also, as with the "alliance" rule, because the "occupancy" rule must check i-vars and their attached number information as contained within lists, it requires high processing costs. Thus, by lowering the priority of this rule's application, there are a lower number of uses of this rule which results in improved computation efficiency.

5.5. **Classifying rules by processing.** When classified by processing, the rules used in this study are classified as follows.

(1) A rule which eliminates an i-var's number candidates from boxes where numbers have not yet been assigned

· candidate elimination rule

(2) Rules which assign numbers to boxes

· unification rule

· list unification rule

(3) Rules which take into account the relationship of the problem's row, column and sub-grid

· comprehension rule

· alliance rule

· occupancy rule

(4) Rules which are used to split the problem state

· splitting rule

· branch cutting rule

The rules in (3) are new rules introduced for the first time under the proposed method. Under the proposed method, to see the multiple relationships among all the rows, columns and sub-grids, the i-var information attached to any given box is transferred to other boxes' i-var information, thus modifying their information. As the amount of information being transferred increases, there is more and more elimination of number candidates for each box. Because of this, more efficient computation is made possible through this study in comparison to the previous study [18] or studies conducted in logic programming [9, 15]. It is believed that the rules shown in (3) will realize more efficient computations for puzzle problems and scheduling problems. Because the previous study [18] utilized AllDifferent atoms and did not take into account the relationships in the problem among the rows, columns and sub-grids, it would not be possible to introduce the three rules in (3).

6. Experiment Result and Discussion.

6.1. **Experiment result.** Experiments were conducted using Prolog, the previous study's method and the proposed method and their processing times were compared. The problems given were five problems with different patterns. Through executing the computation 10 times for each problem, the average processing times were compared. For the processing time for Problem1 and Problem2, Prolog's processing time is represented by proportional ratio at 1. For the processing time for Problem3, Problem4 and Problem5, the processing time under the previous study's method is represented by proportional ratio at 1 because Prolog's processing time is larger compared to other methods (Table 1).

TABLE 1. Processing time

	Prolog	Prior research	Proposed method
Problem1	1	0.063	0.027
Problem2	1	0.020	0.012
Problem3	∞	1	0.259
Problem4	∞	1	0.247
Problem5	∞	1	0.309

6.2. **Considerations.** For Problem1 and Problem2, under the ET computation model method, the information of numbers that can be placed in a box (variable) is controlled by i-vars and some of the i-vars' number information can be transferred to other i-vars or rules during the computation process. Therefore, it is not necessary to process each piece of information found within all i-vars or rules to determine the numbers. Through information transfer between i-vars or between rules, it is possible to greatly improve computation efficiency. Also, compared to logic programming (LP) which executes computation upon successively simplifying problems into the smallest unit (smallest clause), the ET computation model method realizes efficient computation through global processing of a large amount of information without simplifying the problem into the smallest unit. That is, since the information in an AllDifferent atom is more extensive than that found in the smallest unit atoms of LP languages, it is possible to define various constraint processing rules which make good use of that information.

For Problem3, it is confirmed that the "splitting" rule was not used under any of the methods. Computation efficiency is attained through the elimination of unnecessary information and through effective unification. There were 51 unifications executed when using the previous study's method while under the proposed method, there were 19. With the proposed method, because unification appertaining to same number information is executed as a whole, the number of times of unification executed was lowered and as a result it is considered to lead to more efficient computation.

For Problem4, under the previous study's method, the "splitting" rule was executed four times whereas under the proposed method the "splitting" rule was not executed, but the "comprehension" rule was applied, only once, instead. Therefore, the "comprehension" rule, which is unique to the proposed method, made it possible for there to be no need to execute a splitting of the problem state.

For Problem5, the "splitting" rule was used by all methods. However, there were eight occurrences of the "splitting" rule being used with the previous study's method while it was used four times by the proposed method. In addition, the proposed method made effective use of the "comprehension" rule and the "alliance" rule, which resulted in reduced use of problem state splitting. As a result, it made more efficient computation possible.

Also, for Problem3, Problem4 and Problem5, the proposed method makes use of a rule which cannot be used by the previous study's method; it takes into account the relationships among the rows, columns and sub-grids and as a result reduces the number of times that costly problem state splitting must occur, which leads to computation efficiency.

7. **Application of the Proposed Method.** When a constraint satisfaction problem will be simplified to obtain an answer, there are two possible approaches which can be taken: one is to eliminate the i-vars' information and variables with no change to the problem's size as is demonstrated in the proposed method; or one that splits the problem into smaller size units as was demonstrated in the previous method [18]. In cases such as this, where there are two different approaches to consider when solving a problem, more efficient computation can be achieved under the proposed method than the previous method.

Fundamentally, the proposed method is effective for problems for which answers must be obtained by taking into account the relationships among the rows, columns, sub-grids and diagonal lines, making it applicable to puzzle problems such as Pic-a-Pix puzzles, number area problems, crossword puzzles and N Queens problems in addition to the number-place problem used in this paper. It can also be applied to games such as shogi (Japanese chess) and chess.

On the other hand, when it comes to practical applications, the proposed method can be applied to scheduling problems and it is believed that the proposed method would prove to be effective in solving this type of problem [21, 22]. Scheduling problems fall under constraint satisfaction problems and are the ones that determine the optimal order by which to conduct tasks, such as determining which is most efficient or most convenient, out of several tasks to be undertaken. Examples of this type of problem include class scheduling problems, travel scheduling problems and nurse scheduling problems.

8. **Conclusions.** In this study, we proposed a new comprehensive method to effectively solve CSPs and conducted experiments using number-place problems. In particular, we created new rules relevant to the relationship among the rows, columns and sub-grids, something which the previous study did not take into account. And, by skillfully giving the new rules priority, we solved the issues that have been found in the previous study and improved computation efficiency.

Also, by comparing the computation results of Prolog, the previous study's method and the proposed method, we demonstrated the effectiveness of the proposed method.

This time, we conducted our study using number-place problems. However, because the proposed method will be effective for other CSPs, we intend to verify this through other CSPs.

REFERENCES

[1] K. Akama and E. Nantajeewarawat, Formalization of the equivalent transformation computation model, *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol.10, no.3, pp.245-259, 2006.

[2] P. Chippimolchai, K. Akama, T. Ishikawa and V. Wuwongse, Correct computation with multi-head rules in the equivalent transformation framework, *Proc. of the 4th International Conference on Intelligent Technologies*, pp.531-538, 2003.

[3] A. Colmerauer, An introduction to Prolog III, *Communications of the ACM*, vol.33, no.7, pp.69-90, 1990.

[4] R. Dechter, *Constraint Processing*, Morgan Kaufmann Publishers, 2003.

[5] M. Dincbas et al., The constraint logic programming language CHIP, *The 5th Generation Computer Systems*, Tokyo, Japan, 1988.

[6] C. E. Freuder et al., Systematic versus stochastic constraint satisfaction, *IJCAI-95*, pp.2027-2032, 1995.

[7] T. Frühwirth, Theory and practice of constraint handling rules, *Journal of Logic Programming*, vol.37, nos.1-3, pp.95-138, 1998.

[8] J. Jaffar and M. Maher, Constraint logic programming: A survey, *J. of Logic Programming*, vol.19/20, pp.503-581, 1994.

[9] J. W. Lloyd, *Foundations of Logic Programming*, 2nd Edition, Springer-Verlag, 1987.

[10] H. Mabuchi, K. Akama and T. Wakatsuki, Equivalent transformation rules as components of programs, *International Journal of Innovative Computing, Information and Control*, vol.3, no.3, pp.685-696, 2007.

[11] A. Mackworth, Constraint satisfaction, *Encyclopedia of Artifical Intelligence*, vol.1, pp.205-221, 1987.

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

[13] K. Mizuno, H. Kanoh and S. Nishihara, Solving constraint satisfaction problems by an adaptive stochastic search method, *Journal of Information Processing Society of Japan*, vol.39, no.8, pp.2413-2420, 1998.

[14] S. Nishihara, Fundamentals and perspectives of constraint satisfaction problems, *Journal of Japanese Society for Artificial Intelligence*, vol.12, no.3, pp.351-358, 1997.

[15] K. Pettorossi and M. Proietti, Transformation of logic programs: Foundations and techniques, *Journal of Logic Programming*, vol.19/20, pp.261-320, 1994.

[16] E. Tsang, Foundations of constraint satisfaction, *Computation in Cognitive Science*, Academic Press, 1993.

[17] P. van Hentenryck, H. Simonis and M. Dincbas, Constraint satisfaction using constraint logic programming, *Artificial Intelligence*, vol.58, pp.113-159, 1992.
[18] H. Mabuchi, Efficient solution of constraint satisfaction problems by equivalent transformation, *Int. J. of Computational Engineering Research*, vol.3, no.11, pp.61-70, 2013.
[19] K. Miura, K. Akama and H. Mabuchi, Creation of ET rules from logical formulas representing equivalent relations, *International Journal of Innovative Computing, Information and Control*, vol.5, no.2, pp.263-277, 2009.
[20] K. Miura, K. Akama, H. Mabuchi and H. Koike, Theoretical basis for making equivalent transformation rules from logical equivalences for program synthesis, *International Journal of Innovative Computing, Information and Control*, vol.9, no.6, pp.2635-2650, 2013.
[21] A. Allahverdi, C. T. Ng, T. C. E. Cheng and M. Y. Kovalyov, A survey of scheduling problems with setup times or costs, *European Journal of Operational Research*, vol.187, no.3, pp.985-1032, 2008.
[22] P. Brucker, *Scheduling Algorithms*, 5th Edition, Springer, 2006.

Appendix.

- **"candidate elimination 2" rule.** The "candidate elimination 2" rule functions in the same manner as the "candidate elimination" rule from the previous study in that it eliminates information of numbers that were defined within the information attached to an i-var (see Section 4.2). However, with layer atoms, the process for the number assigned by the first argument is different from that for other numbers. If a number that is the same as that assigned by the first argument exists in a row (or a column/sub-grid), the assigned number's information can be eliminated from the i-vars belonging to that row. Also, because rows that do not have number information do not need to be referred to again, it could lead to efficiency in computation. Information for numbers other than assigned ones must be eliminated without reflecting to other i-vars belonging to that row. This is because boxes in any of the rows, columns and sub-grids are referred to only by their assigned numbers and, as such, number information cannot be reflected to boxes (variables) that are not referred to. An example is given below. Here, `*rowi`$(1 \leq$ `i` $\leq 9)$ indicates each individual row.

```
(Layer 3 (3 4 *a~(2 3 4) 5 *b~(2 3 4 5)))
         ((*c~(3 7) 5 *b~(1 3 5))
                ⋮
         *row9)
         *column-list *box-list)
               ↓
(Layer 3 ((*c~(3 7) *b~(1 3 5))
                ⋮
         *row9)
         *column-list *box-list)
```

  The variables `*a` and `*b` which have been eliminated in the example above change to the variables `*a~(2 4)` and `*b~(2 4 5)` respectively. Also, the same process is applied to both `*column-list` and `*box-list`.

- **"variable deletion" rule.** Through its assigned numbers, the proposed method applies a rule based on the information as to which box (variable) these assigned numbers can be placed. Therefore, the applied rule may possibly affect the information of other boxes (variables) and the information attached to i-vars (candidate numbers) may be changed or eliminated. In the proposed method, therefore, a rule to eliminate, from the list, those boxes that do not accept the designated numbers is required. This rule is the "variable deletion" rule. An example follows.

```
(Layer 3 (*a~(2 4)*b~(2 4 5) *c~(3 7) *d~(1 3 5)))
```

```
          (*e~(3 7) 5 *f~(1 3 5))
                ⋮
          *column-list *box-list)
                ↓
(Layer 3 (*c~(3 7) *b~(1 3 5))
                ⋮
          *column-list *box-list)
```

In examining the information of the i-vars `*a` and `*b`, it is seen that the information does not have the number 3. Therefore, in cases where 3 is assigned, both `*a` and `*b` do not need to be referred to because 3 is not contained in their information. As a result, `*a` and `*b` can be eliminated.

- **"unification" rule.** The "unification" rule is the same rule from the previous study (see Section 4.2).
- **"list unification" rule.** Although the concept behind the "list unification" rule in the proposed method is the same as the "number decision" rule from the previous study, its fundamental process is different. In the proposed method, any of the rows, columns and sub-grids are checked in sequence and when it is found that the list length is 1, a rule is applied. A list length that is 1 is defined as a list containing only one i-var. If a list has only one i-var and numbers assigned by a layer atom's first argument are contained within that i-var's information, then those numbers are assigned to that i-var. At the same time, if the assigned numbers do not exist in the i-var information, processing fails. An example follows.

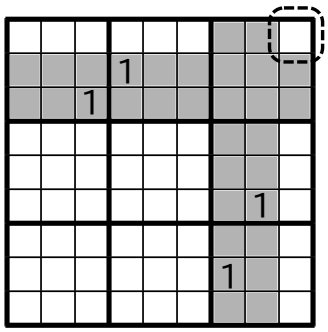

FIGURE 8. "list unification" rule

In Figure 8, only boxes in which the number 1 can be placed are shown. In the box at the upper right that is enclosed with dotted lines, it is clear that only 1 can placed in this box due to the constraints. In the case of a sub-grid (one of the nine sub-grids with a thicker border), such as the one at the upper right, which falls under a layer atom whose first argument is 1, the list which applies to that sub-grid is one which contains only one i-var as shown in the following list. Here, `*boxj`$(1 \leq \mathtt{j} \leq 9)$ indicates each individual sub-grid.

```
(Layer 1 *row-list *column-list
         (*box1
          *box2
          (*a~(1 2 3 4 5 6 7 8 9))
          *box4
             ⋮
          *box9))
```

At this time, because the list which contains the i-var `*a` has only one box to which 1 can be placed, not assigning 1 to `*a` will cause a contradiction due to the constraints. Therefore, the number 1 is assigned to `*a`.

Although the effect produced by this rule is similar to the previous study's "number decision" rule, in the previous study, all number information of every i-var in any lists for any of the rows, columns and sub-grids had to be checked. With the proposed method, however, only lists containing a single i-var must be found and the number information of that i-var needs to be checked, making the computation more efficient.

- **"splitting" rule.** This rule operates under the same process as in the previous study [18]. When problem states arise that cannot be accommodated by the rules described above, this rule splits the problem state into multiple states. However, because this rule can cause increased computation costs, this should be a last resort case, made so by lowering the priority of this rule's application.

- **"branch cutting" rule.** The "branch cutting" rule is one that can lead to computation failure and occurs when there is a "contradiction" in the split state as a result of problem state splitting. In the proposed method, a "contradiction" is defined as cases where no number information has been assigned to a box (variable) and also in cases where the same number has been duplicated within any of the rows, columns and sub-grids. In the case of there being no number information assigned to a box (variable), (Constraint 1) will be violated so computation failure will occur when this happens. Because all assigned numbers must be checked for duplication in the case of there being the same number duplicated within any of the rows, columns and sub-grids, it will increase computation costs. For this, computation is continued until the end and then there is a check to determine whether the final solution satisfies the constraints of the number-place problem. From this, computation efficiency can be realized.