Constraint Satisfaction with a Multi-Dimensional Domain

Masazumi Yoshikawa Shin-ichi Wada
C&C Systems Research Laboratories, NEC Corporation
4-1-1 Miyazaki, Miyamae-ku, Kawasaki 216
JAPAN

Abstract

This paper presents a novel approach to a class of *constraint satisfaction problems (CSPs)*. First, it defines *Multi-Dimensional Constraint Satisfaction Problem (MCSP)*, which is a useful model applicable to many scheduling problems. Second, it proposes an approach for MCSPs. The approach employs both a general problem-solving method and an automatic generation method for problem-solving programs. The problem-solving method is a combined method with backtracking and constraint propagation, based on the features of MCSPs. The automatic generation method analyzes the meaning of constraints and generates a problem-solving program, which is especially efficient for the given problem. Finally, the proposed approach is evaluated by several experiments including scheduling applications and well-known toy problems. Employing both the two methods enables solving a hard MCSP in a reasonable time, merely by describing it in a declarative form.

1 INTRODUCTION

In recent years, a scheduling problem is increasingly being viewed as a *constraint satisfaction problem (CSP)* [Fox 1989]. The CSP has a simple well-defined model and general problem-solving algorithms. Since many scheduling problems can be formulated as CSPs, they can be solved by these algorithms, in a general way.

In order to solve a CSP, two classes of algorithms have been developed. The first group involves *backtracking* algorithms and the second group involves *constraint propagation* algorithms. Backtracking algorithms are guaranteed to solve any CSP, but they suffer from thrashing [Nudel 1983]. On the other hand, several *network-based* constraint propagation algorithms have been developed, using the topology of *constraint networks* [Mackworth 1977, Freuder 1982, Freuder 1990]. They are guaranteed to solve a class of CSPs with a *tree-like* constraint network in polynomial time [Mackworth 1985].

Recently, combined algorithms with backtracking and constraint propagation have been developed [Dechter 1988, Dechter 1990]. They use constraint propagation on tree-like subgraphs of a constraint network in order to reduce the search space for backtracking. They can solve any CSP and are efficient for tree-like constraint networks. They have been the most powerful algorithms to solve CSPs in a general way.

However, even with these algorithms, most scheduling problems are hard problems. This is because, they have a *disjointness* constraint, that prohibits assigning a same *value* (resource) to more than one *variable* (task). Since this constraint is concerned with every combination of two variables, their constraint networks become a *complete graph*, which is the most complex one. In general, a CSP is the more difficult, if the constraint network is the more complex [Zabih 1990]. This is the most critical difficulty for solving scheduling problems in a general way.

In order to overcome this difficulty, the authors took an approach from two points of view:

1. Since there is no algorithm which solves any CSPs efficiently, a problem-solving method, based on the features of the application problems, is required.
2. The network-based algorithms use only the topological features of the constraints. However, focusing on the meaning of constraints, there may be more efficient ways to process them.

This paper presents a novel approach to a class of CSPs. First, it defines *Multi-Dimensional Constraint Satisfaction Problem (MCSP)*, which is a useful model applicable to many scheduling problems. A declarative framework to describe MCSPs is also presented. Second, it proposes an approach for MCSPs. The approach employs a general problem-solving method for MCSPs and an automatic generation method for problem-solving programs.

The problem-solving method is a combined method with backtracking and constraint propagation. It is an efficient method, based on the features of MCSPs. The automatic generation method generates a problem-solving program, which is especially efficient for the given problem. It analyzes constraints in a logical form and selects efficient procedures, according to the meaning of constraints.

Finally, the proposed approach is evaluated by experiments including school curriculum scheduling, production scheduling, work assignment problems, and several well-known MCSP problems.

Employing the two methods enables solving a hard MCSP in a reasonable time, merely by describing it in a declarative form.

The following section defines the MCSP and presents the declarative framework to describe MCSPs. In Section 3, the problem-solving method for MCSPs is proposed. The automatic program generation method is described in Section 4. Section 5 shows and discusses the experimental results. A summary and conclusion are given in Section 6.

2 PROBLEMS

As described in the previous section, most scheduling problems belong to the most difficult class of CSPs. Therefore, the authors focused their attention on a class of CSPs, that is applicable to many scheduling problems. This section defines *Multi-Dimensional Constraint Satisfaction Problem (MCSP)* and presents a declarative framework to describe MCSPs.

2.1 Multi-Dimensional Constraint Satisfaction Problem

A CSP involves a set of N *variables* $v_1, \ldots, v_N$ having *domains* $D_1, \ldots, D_N$, where each D_n defines the set of available values for the variable v_n. An MCSP is a CSP, in which the all domains are same. Namely, $D_1 = D_2 = \ldots = D_N$.

For example, a *Four Color Problem* is an MCSP. It is a problem to assign four colors on every bounded area on a plane, satisfying the constraint that no area has the same color as it's neighboring area. This problem has variables for every area, and a shared domain that is the set of four colors.

Moreover, the domain may be represented by an $I \times J$ array with two (or more) dimensions. Figure 1 illustrates the MCSP variables and domain.

Variables: $\rightarrow n$

variable 1	variable 2	...	variable N

Domain: $\rightarrow j$; $\downarrow i$

value $(1,1)$	value $(1,2)$	...	value $(1,J)$
value $(2,1)$	value $(2,2)$	...	value $(2,J)$
...	...	...	...
value $(I,1)$	value $(I,2)$	...	value (I,J)

Figure 1: Multi-Dimensional Constraint Satisfaction Problem

For example, a school curriculum scheduling problem, by which to assign a teacher and a time for every classroom, is a two-dimensional MCSP. In this case, variables are given classrooms. The domain is represented by an array with two dimensions corresponding to teachers and times. Another example is a production scheduling problem, that consists of N tasks, I production machines, and J time-intervals in a scheduling period.

Many other scheduling problems, such as work assignment problems, can be formulated as MCSPs. Consequently, MCSP is an important subclass of CSPs for scheduling applications.

2.2 DECLARATIVE DESCRIPTION OF MCSPS

This section presents a declarative framework to describe MCSPs. An MCSP consists of a set of variables, a multi-dimensional domain, and a set of constraints.

```
1  (define-set area      area1 area2 area3 area4...)
2  (define-set color     red blue green yellow)
3  (define-set-of-sets neighbors
4    (area1 area2) (area1 area3) (area2 area4)...)
5  (define-constraint exclusive-color
6     ((area color) (area1 color1) (area2 color2))
7    (if (in-same-set-of neighbors area1 area2)
8        (not (= color1 color2))
9        true))
10 (define-problem four-color-problem
11     (:variables area)
12     (:domain (color))
13     (:constraints exclusive-color))
```

Figure 2: Declarative Description of a Four Color Problem

For example, the declarative description of a Four Color Problem is presented in Fig. 2. Lines 1-2 define the sets of variables and a domain. Lines 3-4 define a class used in the constraint definition in lines 5-9. The problem is defined in lines 10-13. The body (lines 7-9) of the constraint definition is a logical form. The meaning of the constraint definition is that:

For every pair of two different assignments (`area color`)s, let the assignments be (`area1 color1`) and (`area2 color2`),
if `area1` and `area2` form a pair of `neighbors`,
`color1` and `color2` must be different,
otherwise OK.

Line 12 defines the shared domain for this MCSP. A Four Color Problem has a one-dimensional domain (a set `color`).

In case of school curriculum scheduling, the domain may have two-dimensions (`teachers` and `time` involved). The problem definition may be as follows:

```
(define-problem school-scheduling
   (:variables classroom)
   (:domain (teacher time)) ;; 2-dimension
   (:constraints ...))
```

The problem is to assign a value, in the two-dimensional domain made up with the sets `teacher` and `time`, to each variable in the set `classroom`, satisfying the all constraints specified in the `:constraints` option.

3 A METHOD TO SOLVE MCSPS

As described in Section 1, most scheduling problems belong to the most difficult class of CSPs. Therefore, the authors developed an efficient method for MCSPs, based on the MCSP features. This section describes two MCSP features and proposes a problem-solving method, based on the features.

3.1 FEATURES OF MCSPS

As mentioned in the preceding section, an MCSP has a multi-dimensional (or single-dimensional) domain that consists of a set of dimensions. In the case of a multi-dimensional domain, the dimensions have independent meanings in the application problem, e.g., teachers and times. Therefore, many constraints refer to only one dimension of the domain.

For example, a school curriculum scheduling problem has the following constraints:

`science-classroom-science-teacher`
A science `teacher` must be assigned to a science `classroom`.

`same-class-different-time`
Different `times` must be assigned to two `classrooms` of the same class.

Constraint `science-classroom-science-teacher` does not refer to the domain dimension `time`, but to the other dimension `teacher`. On the other hand, `same-class-different-time` refers to only `time`.

The constraints, which refer to only one domain dimension, are called *one-dimensional constraints*, while other constraints are called *multi-dimensional constraints*.

Since domain dimensions have independent meanings, most constraints are one-dimensional. This is an important feature of MCSPs. The proposed method is based on this *dimension independence* of MCSPs.

Another MCSP feature is *problem duality*. Since an MCSP has a two (or more) dimensional domain, the problem is assigning a two-dimensional value (i, j) to each variable v_n. Therefore, it can be reformulated into another CSP, in which a (one-dimensional) value i is assigned to a variable vi_n and j is assigned to vj_n. For example, since the school curriculum scheduling problem is assigning a value (`teacher`, `time`) to each `classroom`, it can be reformulated into another CSP with $2N$ variables, N variables vi_n (`classrooms`) for I values (`teachers`) and N variables vj_n (`classrooms`) for J values (`times`). This problem duality is also used in the problem-solving method for MCSPs.

3.2 A PROBLEM-SOLVING METHOD BASED ON MCSP FEATURES

The problem-solving method is based on the MCSP features, dimension independence and problem duality.

The method decomposes an MCSP into three (or more) subproblems, using *problem duality*. The subproblems are:

SP-M A subproblem, which is same as the original MCSP, except that it has only multi-dimensional constraints.

SP-I A subproblem, which corresponds to the dimension i, with N variables vi_n, a domain with size I, and one-dimensional constraints that refer to i.

In the school curriculum scheduling problem, it has N `classrooms` as variables, I `teachers` as values, and one-dimensional constraints that refer to only `teachers`, e.g., `science-classroom-science-teacher`.

SP-J A subproblem, which corresponds to the dimension j, with N variables vj_n, a domain with size J, and one-dimensional constraints that refer to j.

In the example, it has N `classrooms` as variables, J `times` as values, and one-dimensional constraints that refer to only `times`, e.g., `same-class-different-time`.

Then, the method assigns values to variables using a backtracking algorithm and a constraint propagation algorithm, as follows:

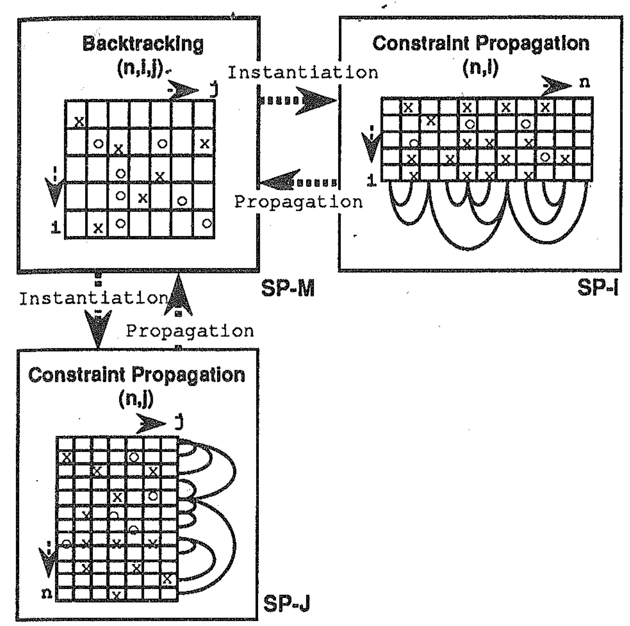

Figure 3: A Problem-Solving Method for MCSPs

1. A backtracking algorithm creates assignment on subproblem SP-M, checking multi-dimensional constraints.
2. A constraint propagation algorithm processes one-dimensional constraints on subproblems SP-I and SP-J.

   In the example, one-dimensional constraints about `times` are propagated in SP-J.

The constraint propagation is triggered when the backtracking assigns a value to a variable. The backtracking selects the candidate values for a variable according to the constraint propagation results. Figure 3 illustrates the problem-solving method.

Here, it should be noticed that the problem-solving method does not specify a certain backtracking algorithm or a constraint propagation algorithm. Existing backtracking algorithms can be joined into this method with small modification. Also, several constraint propagation algorithms can be combined in the problem-solving method. Current experimentation uses a *most-constraint min-conflicts* backtracking algorithm, as described in [Keng 1989], and a naive constraint propagation algorithm AC-3 in [Mackworth 1977].

3.3 DISCUSSION

In order to evaluate the problem-solving method, how existing CSP algorithms work on an MCSP must be considered. Several efficient algorithms have been developed using the topology of *constraint networks*, e.g., [Freuder 1982], [Mackworth 1985], [Dechter 1988], and [Dechter 1990]. They are applicable or efficient with *tree-like* constraint networks. However, most scheduling problems have a disjointness constraint that makes a constraint network form a *complete graph*. Since they belong to the most difficult class of CSPs [Zabih 1990], these algorithms have few merits.

On the other hand, SP-I and SP-J in the proposed problem-solving method have small domain sizes I and J, while the domain size for the original MCSP is $I \times J$. If we propagate e one-dimensional constraint edges on an MCSP, using AC-3, then the complexity is $O(eI^3J^3)$ (See [Dechter 1988]). On the other hand, the complexity on SP-I and SP-J is $O(e_iI^3 + e_jJ^3)$, where e_i and e_j are number of edges on SP-I and SP-J, namely $e = e_i + e_j$. Consequently, the proposed method dramatically decreases the computational time.

Here, it must be considered carefully that SP-I and SP-J have only one-dimensional constraints. Since MCSPs have dimension independence, most constraints are one-dimensional. However, if there are heavy multi-dimensional constraints in an MCSP, the proposed method has few merits. This is the limitation of this method. An example of heavy multi-dimensional constraints is the disjointness constraint, which can be checked in a cheaper manner, using an $I \times J$ array, as described in the next section.

In addition, *the combination of local propagation and backtracking (LPB)* in [Guesgen 1989] is similar to the proposed method, except that it does not use domain dimensions.

4 AN AUTOMATIC PROGRAM GENERATION METHOD

This section describes an automatic program generation method. The method analyzes the meaning of given constraints and generates appropriate procedures to process them. Then, it integrates them into a program to solve the problem. First, a naive program generation method is described. Then, a method to refine a constraint process is proposed.

4.1 A NAIVE PROGRAM GENERATION METHOD

The naive program generation method analyzes a given constraint and generates a constraint process procedure, as follows.

Step 1: A constraint is defined with a logical form. For example, the `exclusive-color` constraint for a Four Color Problem has the following form:

```
(if (in-same-set-of neighbors area1 area2)
    ;; if area1 and area2 is a neighbors pair.
    (not (= color1 color2))
    true)
```

Step 2: What is prohibited by the constraint can be represented by the negation of the logical form. The logical form is negated and normalized into a *conjunctive normal form*:

```
(and (in-same-set-of neighbors area1 area2)
     (= color1 color2))
```

Step 3: The subforms for the conjunctive form (the **and** form) are divided into two sets, *variable-forms* and *value-forms*.

Variable-forms: Subforms, which refer to no domain values (`colors`), but variables (`areas`).

`(in-same-set-of neighbors area1 area2)`

Value-forms: Subforms, which refer to domain values (`colors`), may also refer to variables (`areas`).

`(= color1 color2)`

Step 4: Variable-forms restrict related (combinations of) variables to those which satisfy them. The method generates a procedure which associates the constraint to the related variables, using the variable-forms. In this example, the generated procedure creates constraint edges between all pairs of `neighbors`.

Step 5: Value-forms specify what (combinations of) values are prohibited by the constraint. The method generates a procedure which processes the constraint, using the value-forms. There are three kinds of constraint processes and the category is determined by the problem-solving method, described in Section 3.

The three kinds of constraint processes are, domain-value removal for unary constraints, constraint propagation for one-dimensional binary constraints, and constraint checking for multi-dimensional constraints.

In the case of one-dimensional binary constraints, the value-forms are processed by constraint propagation. The method generates the following propagation procedure:

> **PROC-N**: For each available *value2* (`color2`) value, if there is no available value *value1* (`color1`) such that the *value-form* (`= color1 color2`) is evaluated to be false, remove *value2* (`color2`) value from the domain, otherwise do nothing.

Note that this is the same as the procedure REVISE of AC-3. If an implementation uses another constraint propagation algorithm, this procedure may be modified, according to the algorithm in use.

The constraint procedures, generated by the generation method, are integrated into a problem-solving program, that uses the problem-solving method, described in Section 3. Current experimentation generates a program which preprocesses unary constraints by domain-value removal, propagates one-dimensional binary constraints on subproblems by AC-3, and checks multi-dimensional constraints in a most-constraint min-conflicts backtracking algorithm.

4.2 A CONSTRAINT PROCESS REFINEMENT METHOD

As shown in the naive method, *value-forms* are used for three kinds of constraint processes. In any case, they are evaluated with certain values to determine whether or not these values satisfy the constraint. Therefore, the naive method repeats the evaluation many times. On the other hand, considering the meaning of value-forms, there are more efficient ways to process a constraint.

4.2.1 Local Refinement Method

The complexity of the naive procedure PROC-N is $O(I^2)$, where I is the domain size. In the `exclusive-color` example, since the value-form (`= color1 color2`) prohibits `color1` and `color2` from taking the same value, it can be propagated as follows:

> **PROC-1**: Check whether the available value for *value1* (`color1`) is unique or not. If unique, remove the unique value from the domain of *value2* (`color2`), otherwise do nothing.

Using this procedure, the constraint can be propagated in constant time when an implementation provides a domain size counter for every variable.

For another example, a job-shop production scheduling problem has a constraint that specifies the ordering among two tasks. The value-form of this constraint may be (`not (< time1 time2)`). An $O(I)$ procedure, to propagate the constraint, is:

> **PROC-2**: Find the minimum available value of *value1* (`time1`), remove the values less-than-or-equal-to the minimum value from the domain of *value2* (`time2`).

As shown in the examples, several common value-forms have an efficient procedure to process them. The refinement method provides such efficient procedures associated with a pattern for a value-form, such as (= *value1 value2*). The refinement method takes matching between a given value-form and provided patterns. If a matching pattern is found, then the associated procedure is used in place of the naive procedure. They are provided separately, according to the three kinds of constraint processing.

Here, it should be noticed that procedures PROC-1 and PROC-2 depend on propagation algorithm AC-3. These procedure must be modified, corresponding to the propagation algorithm in use.

4.2.2 Global Refinement Method

The method described above refines a propagation process for one constraint edge. On the other hand, it is possible to refine the propagation process for a set of constraint edges into an efficient procedure. This refinement is accomplished in almost the same manner, but it uses variable-forms, as well as value-forms.

Consider the `same-class-different-time` constraint for the school curriculum scheduling in Section 3. It has a value-form and a variable- form as follows:

Variable-form:

```
(in-same-set-of classrooms-for-the-same-class
                classroom1 classroom2)
```

Value-form:

```
(= time1 time2)
```

The constraint propagation from one variable to all the other variables in a `classrooms-for-the-same-class` set is refined into a procedure:

> **PROC-3**: Check whether or not the available *value1* (`time1`) value is unique. If unique, remove the unique value from the domains for all the other variables in a *set* (`classrooms-for-the-same-class`), otherwise do nothing.

This refinement reduces the complexity from $O(mI^2)$ into $O(m)$, where m is the size of a *set*.

As described in Section 3, thorough checking of a disjointness constraint has large costs in the problem-solving method. The constraint checking can be replaced by the following procedure.

> **PROC-4**: Provide an $I \times J$ Boolean array that expresses the domain. Look up an array entry in order to check the disjointness constraint. When backtracking assigns a value, mark a corresponding array entry in order to specify that the value is unavailable.

If there were no backtrack to assign values to all variables, PROC-4 reduces the total checking cost from $O(N^3)$ into $O(N^2)$. This refinement is more effective in general cases.

4.3 DISCUSSION

The effectiveness of the refinement method has already been shown. Here, the novelty and limitation for this method are discussed.

Guesgen's CONSAT also provides a constraint description language and a constraint compiler, which improves constraint propagation processes [Guesgen 1989]. The compiler eliminates checking variables, which has no relation to a given constraint. However, it does not refine the propagation process with the related variables. The improvement of CONSAT compiler is similar to Step 4 of the naive program generation method in Section 4.1.

The most close research to the refinement method is an arc consistency algorithm AC-5 in [Deville 1991]. AC-5 uses the feature of *functional* and *monotonic* constraints in order to reduce the complexity. It is based on almost the same idea as the *local* refinement method for constraint propagation, except that it does not handle *disequation* constraints, such as disjointness. Moreover, AC-5 does not include the *global* refinement method or refinements for domain-value removal and constraint checking.

It is trivial that this refinement method is effective only when a value-form matches a provided pattern. This is the limitations of the method.

5 EXPERIMENTAL RESULTS

This section evaluates the proposed approach with several experiments including school curriculum scheduling, production scheduling, work assignment, and several well-known CSP problems (See Appendix).

The computational times used for a school curriculum scheduling and a production scheduling problem are shown in Table 1. For each problem, both a two-dimensional and a one-dimensional formalization are examined. A one-dimensional formalization represents the same problem as a two-dimensional one, except that the two-dimensional domain is elongated into a one-dimensional domain. This elongation causes that the proposed problem-solving method works in the same way as LPB in [Guesgen 1989] works (See Section 3.3). Therefore, a comparison between one-dimensional and two-dimensional formalization shows the improvement by using domain dimensions. Here, the constraint process refinement method, proposed in Section 4, is not used, except PROC-4 for the disjointness constraint. [1]

In the case of Problem A, using a multi-dimensional domain causes almost 80 times the previous efficiency. This marked result indicates the great effect of the proposed problem-solving method. On the other hand, the ratio is 1.78 for Problem B. This is because that Problem B is smaller than Problem A. As discussed in Section 3, the complexity of propagating one-dimensional constraints is $O(eI^3J^3)$ vs. $O(e_iI^3 + e_jJ^3)$. Therefore, the method is more effective for a larger problem, namely larger values e, I, and J. Consequently, the problem-solving method is more effective for a larger and more tightly constrained problem.

[1] This is because the one-dimensional curriculum scheduling problem (Problem A') can not be solved in four days, without PROC-4.

Table 1: Experimental Results of the Problem-Solving Method

Problem		a. Two-dims. (seconds)	b. One-dim. (seconds)	Ratio (b/a)
A,A'	Curriculum Scheduling	1,921	151,900	79.04
B,B'	Production Scheduling	3.94	7.01	1.78

Table 2: Experimental Results of the Constraint Process Refinement Method

Problem		a. Refined (seconds)	b. Naive (seconds)	Ratio (b/a)	Procedures
A	Curriculum Scheduling	866.60	30400	35.07	PROC-2, 3, 4, etc.
B	Production Scheduling	3.73	47.71	12.79	PROC-2, 4, etc.
C	Work Assignment	82.48	87.62	1.06	PROC-1, 3, etc.
D	Work-Pattern Assignment	1.96	2.38	1.22	PROC-1, etc.
E	50-Queen	148.20	1110	7.49	PROC-4, special proc.
F	Four Color Problem	217.60	598.60	2.80	PROC-1.
G	Four Color Problem (no solution)	0.42	10.34	24.60	PROC-1.
H	Zebra Problem	0.41	0.60	1.44	PROC-3, etc.

Table 2 compares the computational times for the same problem in two cases: one is when the constraint process refinement method is applied, and only the naive program generation method is used in the other case. [2] These results for each problem are taken from exactly the same problem definition. The procedures used in the refinement method are also listed in the table.

One of the most remarkable results is that the PROC-4, which refines checking the disjointness constraint, has a great advantage (in Problem A, B, and E). Comparing the results of D and H, since PROC-3 (in Problem H) is a global refinement procedure, the gains are larger than a local refinement procedure PROC-1 in Problem D. As discussed in Section 4.3, AC-5 does not include the refinement of constraint checking and global refinement method. Consequently, the proposed refinement method is more effective than merely employing AC-5.

Consider the Four Color Problems (F and G). Since the unique constraint is refined by PROC-1, the gains for the refinement method are comparatively large. In the case of Problem G, an arc consistency algorithm as AC-3 proves that it has no solution without any backtracking. Namely, almost all the time is spent in constraint propagation. Therefore, the refinement method causes a high efficiency.

In addition, it should be mentioned that the same procedures are used in several problems involving different application fields. This means that the constraint process refinement method is applicable for many applications.

Although the constraint process refinement method refines only a part of constraint processes, it has considerable merits for many application problems. Moreover, it is very effective especially for hardly constrained problems. Consequently, two proposed methods enable solving a hard problem in a dramatically efficient way.

[2] Current experimentation uses a most-constraint min-conflicts backtracking algorithm and a naive constraint propagation algorithm AC-3.

6 CONCLUSION

In this paper, a novel approach to a class of CSPs is presented. First, it defines *Multi-Dimensional Constraint Satisfaction Problem (MCSP)*, which is a useful model applicable to many scheduling problems. Second, it proposes an approach for MCSPs, focusing on the features of MCSPs and the meaning of constraints. The approach employs a general problem-solving method for MCSPs and an automatic generation method of problem-solving programs. Finally, the proposed approach is evaluated by several experiments including scheduling applications and well-known toy problems. Employing these methods enables solving a hard MCSP in a reasonable time, merely by describing it in a declarative form.

Appendix: Test Problems

A *A school curriculum scheduling problem* is presented as an example. It has 160 variables (32 classrooms for every 5 classes) and 352 values (11 teachers × 32 times). The constraints are, `science-classroom-science-teacher`, `same-class-different-time`, `disjointness`, a constraint specifying continuous classrooms, a constraint specifies that classrooms in the same class and the same subjects must not be assigned in the same day, etc.

A' The same problem as **A**, except that the two-dimensional domain (11 × 32) is elongated into a one-dimensional domain (352).

B *A job-shop production scheduling problem* with 44 variables (tasks), 6 production machines × 10 time-intervals. It is a very simple test problem developed for experimental purpose. It includes, specification regarding off-days and scheduled machine maintenance, relation among a task and a machine, due dates, task orderings, and a few constraints.

B' The same problem as **B**, except that the two-dimensional domain (6 × 10) is elongated into a one-dimensional domain (60).

C *A work assignment problem* with 114 variables (works) and one-dimensional domain (21 workers). Since the time of a work is given, there are no time dimension for the domain. Constraints are, exclusive assignments for the same work time, specification of workers' available times, license for workers, standard working time length, etc.

D The same problem as **C**, except that the 114 works are preprocessed and combined into 22 work-patterns.

E *N-Queen problems.* They have N variables (rows) and a one-dimensional domain with N values (columns).

F *A Four Color Problem* with 560 variables (areas), 4 values (colors), and 1583 pairs of neighboring areas. It is a one-dimensional problem.

G *A Four Color Problem.* It has 152 variables (areas) and 413 neighborings. Since the available set of colors for each area is restricted, it has no solution.

H *Zebra Problem* in [Dechter 1990]. It is a one-dimensional problem with 25 variables (5 cigarettes, 5 pets, 5 persons, 5 houses, and 5 drinks) and 5 values (positions). It has only one solution.

Acknowledgments

The authors would like to express their thanks to Tatsuo Ishiguro, Masahiro Yamamoto, Takeshi Yoshimura, Masanobu Watanabe and Tomoyuki Fujita for their encouragement in this work. Further, they also thank Yoshiyuki Koseki for his valuable advice.

References

[Dechter 1988] R. Dechter and J. Pearl, "Network-Based Heuristics for Constraint-Satisfaction Problems", *Artificial Intelligence*, Vol. 34, 1988, pp. 1-38.

[Dechter 1990] R. Dechter, "Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition", *Artificial Intelligence*, Vol. 41, 1990, pp. 273-312.

[Deville 1991] Y. Deville and P.V. Hentenryck, "An Efficient Arc Consistency Algorithm for a Class of CSP Problems", *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, 1991, pp. 325-330.

[Fox 1989] M.S. Fox, N. Sadeh, and C. Baykan, "Constrained Heuristic Search", *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, pp. 309-315.

[Freuder 1982] E.C. Freuder, "A Sufficient Condition for Backtrack-Free Search", *Journal of the Association for Computing Machinery*, Vol. 29, No. 1, January 1982, pp.24-32.

[Freuder 1990] E.C. Freuder, "Complexity of K-Tree Structured Constraint Satisfaction Problems", In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1990, pp.4-9.

[Guesgen 1989] H.W. Guesgen, "A Universal Constraint Programming Language", In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, pp. 60-65.

[Keng 1989] N. Keng and D.Y.Y. Yun, "A Planning/Scheduling Methodology for the Constrained Resource Problem", In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 1989, pp. 998-1003.

[Mackworth 1977] A.K. Mackworth, "Consistency in networks of relations", *Artificial Intelligence*, Vol. 8, 1977, pp. 99-118.

[Mackworth 1985] A.K. Mackworth and E.C. Freuder, "The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems", *Artificial Intelligence*, Vol. 25, 1985, pp. 65-74.

[Nudel 1983] B. Nudel, "Consistent-Labeling Problems and their Algorithms: Expected-Complexities and Theory-Based Heuristics", *Artificial Intelligence*, Vol. 21, 1983, pp. 135-178.

[Zabih 1990] R. Zabih, "Some Applications of Graph Bandwidth to Constraint Satisfaction Problems", In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1990, pp.46-51.