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 1988]. On the other hand, several network-based constraint propagation algorithms have been developed, using the topology of constraint networks [Macworth 1977, Freuder 1982, Freuder 1990]. They are guaranteed to solve a class of CSPs with a tree-like constraint network in polynomial time [Macworth 1985]. Recently, combined algorithms with backtracking and constraint propagation have been developed [Dechter 1989, 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 more difficult, if the constraint network is the more complex [Zabin 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 v1, ..., vN having domains D1, ..., DN, where each Di defines the set of available values for the variable vi. An MCSP is a CSP, in which the all domains are same. Namely, Dr = D2 = ... = DN.

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 x J array with two (or more) dimensions. Figure 1 illustrates the MCSP variables and domain.

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.

For example, the declarative description of a Four Color Problem is presented in Fig. 2. Lines 1-2 define the data structure 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 (area1 and color1), 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 single-dimensional 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, single-constituent constraint 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 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.

Constrains 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 N-) dimensional domain, the problem is assigning a two-dimensional value (i,j) to each variable vij. Therefore, it can be reformulated into a two-variable CSP, in which a one-dimensional value i is assigned to a variable vij and j is assigned to vji.

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, V variable vij (classroom) for i values (teachers) and N variable vij (classroom) for j values (time). 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 the same as the original MCSP, except that it has only one-dimensional constraints.
- **SP-I**: A subproblem, which corresponds to the dimension i, with N variables vij; a domain with size I, and one-dimensional constraints that refer to i.
- **SP-J**: A subproblem, which corresponds to the dimension j, with N variables vij, a domain with size J, and one-dimensional constraints that refer to j.

In each subproblem, 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 methods does not specify a certain backtracking algorithm or a constraint propagation algorithm. Existing backtracking algorithms can be combined into this method with small modification. Also, several constraint propagation algorithms can be combined in the problem-solving method. Current experiments use a most-constant 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 topological of constraint networks, e.g., [Freyd 1982], [Mackworth 1985, 1989], and [Dechter 1989]. They are applicable or efficient with trees-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 [Zahor 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 x J. If we propagate a one-dimensional constraint on edges of SP-M, using AC-3, then the complexity is $O(I^2P)$ (See [Dechter 1988]). On the other hand, the complexity on SP-I and SP-J is $O(e_i^2 + e_j^3)$, 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 x J array, described in the next section.

In addition, the combination of local propagation and backtracking (LPAB) [Gusen 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 naïve program generation method is described. Then, a method to refine a constraint process is proposed.

4.1 A NAÏVE PROGRAM GENERATION METHOD

The naïve program generation method analyses 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 area area2)
if area and area2 is a neighboring pair
then (color color2)
true

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 combined into this method with small modification. Also, several constraint propagation algorithms can be combined in the problem-solving method. Current experiments use a most-constant min-conflicts backtracking algorithm, as described in [Keng 1989], and a naive constraint propagation algorithm AC-3 in [Mackworth 1977].
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:

\[
\text{(and (in-same-set-of-neighbors area1 area2) (\text{color1}) (\text{color2}))}
\]

Step 3: The subforms for the conjunctive form (the and form in Section 2) are divided into two sets, variable-forms and value-forms:

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

\[
\text{(in-same-set-of-neighbors area1 area2)}
\]

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

\[
(= \text{color1} \text{color2})
\]

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:

1. Domain-value removal for unary constraints, constraint propagation for one-dimensional binary constraints, and constraint propagation 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 value \( v \) (color2) value, if there is no available value \( v \) (color1) such that the value-form \((= \text{color1} \text{color2})\) is evaluated to be false, remove \( v \) (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 2. The present experiment implementation generates a program which preprocesses unary constraints by domain-value removal, propagates one-dimensional binary constraints on subproblems by AC-1, and checks multi-dimensional constraints in a most-constraint-min-conflits backtracking algorithm.

### 4.2 A CONSTRAINT PROCESS REFINEMENT METHOD

**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 \((= \text{color1} \text{color2})\) prohibits \( \text{color1} \) and \( \text{color2} \) from taking the same value, it can be propagated as follows:

**PROC-1:** Check whether the available value \( v \) (color1) is unique or not. If unique, remove the unique value from the domain of \( v \) (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 \((\preceq \text{time1} \text{time2})\). An \( O(I) \) procedure, to propagate the constraint, is:

**PROC-2:** Find the minimum available value \( v \) (time1), and the minimum value from the domain of \( v \) (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 \((= \text{color1} \text{color2})\). 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 procedure **PROC-1** is independent on the propagation algorithm AC-1. These procedures 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 job-shop scheduling problem in Section 3. It has a value-form and a variable-form as follows:

**Variable-form:**

\[
(= \text{class1} \text{class2})
\]

**Value-form:**

\[
(\preceq \text{time1} \text{time2})
\]

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

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

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

As described in Section 3, thorough checking of a disjunctive 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 enters the domain. Look up an array entry in order to check the disjunctive constraint. When backtracking assigns a value, mark a corresponding entry array 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 \( O(m^2) \) into \( O(mN^2) \). 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.

Gueugen's CONSAT also provides a constraint description language and a constraint compiler, which improves constraint propagation processes (Gueugen 1980). The method eliminates generating 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 (Dechter 1991). AC-5 uses the feature of functional and monotonic constraints in order to reduce the complexity. It is based on the same idea as the proposed method for constraint propagation, except that it does not handle disjunctive 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 formulation are examined. A one-dimensional formulation represents the same problem as a two-dimensional one, except that the two-dimensional domain is elobated into a one-dimensional domain. This elobation causes that the proposed problem-solving method works in the same way as LPB in (Gueugen 1980) (See Section 3.3). Therefore, a comparison between one-dimensional and two-dimensional formulation shows the improvement by using domain dimensions. Here, the constraint problem refinement method, proposed in Section 4, is not used, except PROC-4 for the disjointness constraint.

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 \( \Theta(e^2 I^2 N) \) vs. \( \Theta(e^2 I + e^2 I^2) \). Therefore, the method is more effective for a larger problem, namely larger values of \( I \) and \( e \). Consequently, the problem-solving method is more effective for a larger and more tightly constrained problem.
Table 1: Experimental Results of the Problem-Solving Method

<table>
<thead>
<tr>
<th>Problem</th>
<th>a. Two-dims. (seconds)</th>
<th>b. One-dim. (seconds)</th>
<th>Ratio (b/a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, A'</td>
<td>1,921</td>
<td>151,930</td>
<td>78.94</td>
</tr>
<tr>
<td>B, B'</td>
<td>2.94</td>
<td>7.01</td>
<td>1.78</td>
</tr>
</tbody>
</table>

Table 2: Experimental Results of the Constraint Process Refinement Method

<table>
<thead>
<tr>
<th>Problem</th>
<th>a. Refined (seconds)</th>
<th>b. Naive (seconds)</th>
<th>Ratio (b/a)</th>
<th>Procedures</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Curriculum Scheduling</td>
<td>366.60</td>
<td>344.40</td>
<td>1.06</td>
<td>PROC-2, 3, 4, etc.</td>
</tr>
<tr>
<td>B Production Scheduling</td>
<td>3.73</td>
<td>47.71</td>
<td>12.79</td>
<td>PROC-2, 4, etc.</td>
</tr>
<tr>
<td>C Work Assignment</td>
<td>82.48</td>
<td>87.62</td>
<td>1.06</td>
<td>PROC-1, 3, etc.</td>
</tr>
<tr>
<td>D Work-Pattern Assignment</td>
<td>1.26</td>
<td>2.38</td>
<td>1.22</td>
<td>PROC-1, etc.</td>
</tr>
<tr>
<td>E Sf-Queue</td>
<td>148.29</td>
<td>1110</td>
<td>7.49</td>
<td>PROC-4, specialized proc.</td>
</tr>
<tr>
<td>F Four Color Problem</td>
<td>327.60</td>
<td>598.60</td>
<td>1.80</td>
<td>PROC-1, etc.</td>
</tr>
<tr>
<td>G Four Color Problem</td>
<td>0.42</td>
<td>10.34</td>
<td>24.60</td>
<td>PROC-1, etc.</td>
</tr>
<tr>
<td>H Zebra Problem</td>
<td>0.41</td>
<td>0.60</td>
<td>1.44</td>
<td>PROC-3, etc.</td>
</tr>
</tbody>
</table>

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. These results show that the problem is 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 D). Comparing the results of D and II, since PROC-4 (in Problem II) 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-4, 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.

In summary, a constraint satisfaction problem has 160 variables (12 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 specifying 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 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 ordering, and a few constraints.
B' The same problem as B, except that the two-dimensional domain (6 × 10) is elongated into one-dimensional domain (60).

A work assignment problem with 114 variables (workers) and one-dimensional domain (21 workers).

C A work assignment problem with 114 variables (workers) 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 1565 pairs of neighboring areas. It is a one-dimensional problem.
G A Four Color Problem. It has 192 variables (areas) and 413 neighborhoods. 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 horses, and 5 drinks) and 3 values (positions). It has only one solution.

Appendix: Test Problems

A A school curriculum scheduling problem is presented as an example. It has 190 variables (12 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 specifying 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 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 ordering, and a few constraints.

References

