Optimizing with constraints: a case study in scheduling maintenance of electric power units

Daniel Frost and Rina Dechter*
Dept. of Information and Computer Science,
University of California, Irvine, CA 92717-3425
{frost, dechter}@ics.uci.edu

Abstract

A well-studied problem in the electric power industry is that of optimally scheduling preventative maintenance of power generating units within a power plant. We show how these problems can be cast as constraint satisfaction problems and provide an "iterative learning" algorithm which solves the problem in the following manner. In order to find an optimal schedule, the algorithm solves a series of CSPs with successively tighter cost-bound constraints. For the solution of each problem in the series we use constraint learning, which involves recording additional constraints that are uncovered during search. However, instead of solving each problem independently, after a problem is solved successfully with a certain cost-bound, the new constraints recorded by learning are used in subsequent attempts to find a schedule with a lower cost-bound.

We show empirically that on a class of randomly generated maintenance scheduling problems iterative learning reduces the time to find a good schedule. We also provide a comparative study of the most competitive CSP algorithms on the maintenance scheduling benchmark.

1 Introduction

This paper focuses on a well-studied problem of the electric power industry: optimally scheduling preventative maintenance of power generating units within a power plant. We define a formal model which captures most of the interesting characteristics of these problems, cast the model as a constraint satisfaction problem (CSP), and then evaluate some of the most powerful constraint solving algorithms for its solution.

*The authors thank the Electric Power Research Institute for its support through grant RP 8014-06.

A typical power plant consists of one or two dozen power generating units which can be individually scheduled for preventive maintenance. Both the required duration of each unit's maintenance and a reasonably accurate estimate of the power demand that the plant will be required to meet throughout the planning period are known in advance. The general purpose of determining a *maintenance schedule* is to determine the duration and sequence of outages of power generating units over a given time period, while minimizing operating and maintenance costs over the planning period, subject to various constraints. A maintenance schedule is often prepared in advance for a year at a time, and scheduling is done most frequently on a week-by-week basis. The power industry generally considers shorter term scheduling, up to a period of one or two weeks into the future, to be a separate problem called "unit commitment."

Computational approaches to maintenance scheduling have been intensively studied since the mid 1970's. Dopazo and Merrill [5] formulated the maintenance scheduling problem as a 0-1 integer linear program. Zurm and Quintana [18] used a dynamic programming approach. Egan [6] studied a branch and bound technique. More recently, techniques such as simulated annealing, artificial neural networks, genetic algorithms, and tabu search have been applied [13].

We propose an approach to maintenance scheduling based on the constraint satisfaction problem framework [4]. In this model, there are a finite number of variables, and associated with each variable is a finite domain of values. A solution to a CSP assigns to each variable a value from its domain, subject to a set of constraints that specify that some combinations of assignments are not allowed. Algorithms for CSPs usually find one or more solutions, or report that no solution exists. Many CSP search algorithms are based on backtracking, or depth-first, search. The general constraint satisfaction problem is NP-complete.

We report the results of applying some of the most powerful constraint processing techniques developed in recent years [9, 8, 10, 3, 16] to the maintenance scheduling problem. Most of the empirical evaluation of constraint algorithms was done with purely random binary CSPs. Applying the algorithms to maintenance scheduling-based CSPs (MSCSPs) provides a testbed of problem instances that have an interesting structure and non-binary constraints. Our preliminary empirical results indicate that algorithms which are superior on random uniform binary CSPs are also superior on maintenance scheduling problems, thus providing some validation to the empirical approach based on pure random problems.

The constraint framework consists entirely of so-called *hard* constraints, those which must be satisfied for a solution to be valid. Optimization problems can be viewed as having also *soft* constraints, which can be partially satisfied. The problem then is to find the "best" partially satisfied solution. To avoid explicit soft constraints, or objective functions as they are called in the Operations Research literature, we approach optimization as solving a series of related CSPs, each consisting solely of hard constraints. The CSPs in the series differ in that a hard constraint (or group of constraints) corresponding to the objective function with a particular cost-bound is tighter in each succeeding problem in the series. The tighter constraints result from a reduced cost-bound in the function being optimized. An optimal solution is found by determining the lowest cost-bound for which the corresponding constraint satisfaction problem has a solution. A similar approach was used recently to find a shortest plan using satisfiability and CSP techniques [12, 2].

We present experiments with five algorithms that have proven most useful when tested on random problems. In general, when an algorithm is applied to a maintenance problem instance, it solves each of the corresponding CSPs independently. For the new "iterative learning" procedure, an algorithm that learns new constraints during the search is used, and constraints learned during one instance of the series are applied on later instances. This approach was particularly beneficial for the optimization task.

2 Maintenance Scheduling Problem Description

As a problem for an electric power plant operator, maintenance scheduling must take into consideration such complexities as local holidays, weather patterns, constraints on suppliers and contractors, national and local laws and regulations, and other factors that are germane only to a particular power plant. Our simplified model is similar to those appearing in most scholarly articles, and follows closely the approach of Yellen and his co-authors [1, 17]. The maintenance scheduling problem can be represented by a rectangular matrix (see Fig. 1). Each entry in the matrix represents the status of one generating unit for one week. We will use the terms week and time period interchangeably. A unit can be in one of three states: ON, OFF, or MAINT.

2.1 Parameters

A specific maintenance scheduling problem, in our formulation, is defined by a set of parameters, which are listed in Fig. 2. Parameters U, the number of units, and W, the number of weeks, control the size of the schedule. Many power plants have a fixed number of crews which are available to carry out the maintenance; therefore the parameter M specifies the maximum number of units which can be undergoing maintenance at any one time.

In this paragraph and elsewhere in the paper we adopt the convention of quantifying the subscript i over the number of units, $1 \leq i \leq U$, and the subscript t over the number of weeks, $1 \leq t \leq W$. Several parameters specify the characteristics of the power generating units. The costs involved in preventative maintenance, m_{it}, can vary from unit to unit and from week to week; for instance, hydroelectric units are cheaper to maintain during periods of low water flow. The predicted operating cost of unit i in week t is given by c_{it}. This quantity varies by type of unit and also in

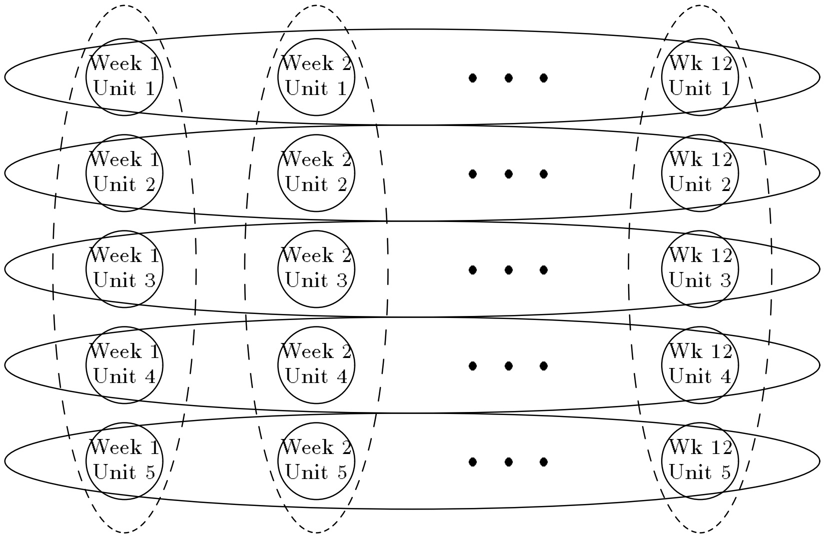

Figure 1: A diagrammatic representation of a maintenance scheduling constraint satisfaction problem. Each circle stands for a variable representing the status of one unit in one week. The dashed vertical ovals indicate constraints between all of the units in one week: meeting the minimum power demand and optimizing the cost per week. The horizontal ovals represent constraints on one unit over the entire period: scheduling an adequate period for maintenance.

response to fuel costs. For example, the fuel costs of nuclear units are low and change little over the year, while oil-fired units are typically more expensive to operate in the winter, when oil prices often increase.

Parameter k_i specifies the maximum power output of unit i. Most formulations of maintenance scheduling consider this quantity constant over time, although in reality it can fluctuate, particularly for hydro-electric units.

The permissible window for scheduling the maintenance of a unit is controlled by parameters e_i, the earliest starting time, and l_i, the latest allowed starting time. These parameters are often not utilized (that is, e_i is set to 1 and l_i is set to W) because maintenance can be performed at any time. The duration of maintenance is specified by parameter d_i.

Sometimes the maintenance of two particular units cannot be allowed to overlap, since they both require a particular unique resource, perhaps a piece of equipment or a highly trained crew member. Such incompatible pairs of units are specified in the set $N = \{(i_1, i_2), \ldots, (i_{n-1}, i_n)\}$.

The final input parameter, D_t, is the predicted power demand on the plant in each week t. The parameters x_{it} are the output of the scheduling procedure, and define the maintenance schedule. x_{it} can take on one of three values:

- ON: unit i is on for week t, can deliver k_i power for the week, and will cost c_{it}

	Input:
U	number of power generating units
W	number of weeks to be scheduled
M	maximum number of units which can be maintained simultaneously
m_{it}	cost of maintaining unit i in period t
c_{it}	operating cost of unit i in period t
k_i	power output capacity of unit i
e_i	earliest maintenance start time for unit i
l_i	latest maintenance start time for unit i
d_i	duration of maintenance for unit i
N	set of pairs of units which cannot be maintained simultaneously
D_t	energy (output) demand in period t
	Output:
x_{it}	status of unit i in period t: ON, OFF or MAINT

Figure 2: Parameters which define a specific maintenance scheduling problem.

to run;

- OFF: unit i is off for week t, will deliver no power and will not result in any cost;
- MAINT: unit i is being maintained for week t, will deliver no power, and will cost m_{it}.

2.2 Constraints

A valid maintenance schedule must meet the following constraints or domain requirements, which arise naturally from the definition and intent of the parameters.

First, the schedule must permit the overall power demand of the plant to be met for each week. Thus the sum of the power output capacity of all units not scheduled for maintenance must be greater than the predicted demand, for each week. Let $z_{it} = 1$ if $x_{it} =$ ON, and 0 otherwise. Then the schedule must satisfy the following inequalities.

$$\sum_i z_{it} k_i \geq D_t \quad \text{for each time period } t \tag{1}$$

The second constraint is that maintenance must start and be completed within the prescribed window, and the single maintenance period must be continuous, uninterrupted, and of the desired length. The following conditions must hold true for each unit i.

$$\text{(start)} \quad \text{if } t < e_i \text{ then } x_{it} \neq \text{MAINT} \tag{2}$$

$$\text{(end)} \quad \text{if } t \geq l_i + d_i \text{ then } x_{it} \neq \text{MAINT} \tag{3}$$

$$\text{(continuous)} \quad \text{if } x_{it_1} = \text{MAINT and } x_{it_2} = \text{MAINT and } t_1 < t_2$$

$$\text{then for all } t, t_1 < t < t_2,\ x_{it} = \text{MAINT} \tag{4}$$

$$\text{(length)} \quad \text{if } t_1 = \min_t(x_{it} = \text{MAINT}) \text{ and } t_2 = \max_t(x_{it} = \text{MAINT})$$

$$\text{then } t_2 - t_1 + 1 = d_i \tag{5}$$

$$\text{(existence)} \quad \exists t \text{ such that } x_{it} = \text{MAINT} \tag{6}$$

The third constraint is that no more than M units can be scheduled for maintenance simultaneously. Let $y_{it} = 1$ if $x_{it} =$ MAINT, and 0 otherwise.

$$\sum_i y_{it} \leq M \quad \text{for each time period } t \tag{7}$$

The final constraint on a maintenance schedule is that incompatible pairs of units cannot be scheduled for simultaneous maintenance.

$$\text{if } (i_1, i_2) \in N \text{ and } x_{i_1 t} = \text{MAINT then } x_{i_2 t} \neq \text{MAINT} \quad \text{for each time period } t \tag{8}$$

After meeting the above constraints, we want to find a schedule which minimizes the maintenance and operating costs during the planning period. Let $w_{it} = m_{it}$ if $x_{it} =$ MAINT, c_{it} if $x_{it} =$ ON, and 0 if $x_{it} =$ OFF.

$$\text{Minimize} \quad \sum_i \sum_t w_{it} \tag{9}$$

Objective functions other than (9) can also be used. For example, it may be necessary to reschedule the projected maintenance midway through the planning period. In this case, a new schedule which is as close as possible to the previous schedule may be desired, even if such a schedule does not have a minimal cost.

3 Formalizing Maintenance Problems as CSPs

There are several ways to encode the maintenance scheduling problem in the constraint satisfaction framework. The formulation involves a tradeoff between the number of variables, the number of values per variable, and the constraints arity.

We defined the problem's output variables as variables in the CSP, and specified the problem's constraints in a relational manner in order to allow our general purpose CSP algorithms to be applied with minimal modification.

We encode maintenance scheduling problems as CSPs with $3 \times U \times W$ variables. The variables can be divided into a set of $U \times W$ visible variables, and two $U \times W$

size sets which we call hidden variables. Each variable has two or three values in its domain. Both binary and higher arity constraints appear in the problem. The visible variables X_{it} correspond directly to the output parameters x_{it} of the problem definition, having the corresponding domain values {ON, OFF, MAINT}.

The first set of hidden variables, Y_{it}, signifies the maintenance status of unit i during week t. The domain of each Y variable is {FIRST, SUBSEQUENT, NOT}. Y_{it} = FIRST indicates that week t is the beginning of unit i's maintenance period. Y_{it} = SUBSEQUENT indicates that unit i is scheduled for maintenance during week t and for at least one prior week. Y_{it} = NOT, indicates no maintenance. Binary constraints between each X_{it} and Y_{it} are required to keep the two variables synchronized (we list the compatible value combinations):

X_{it}	Y_{it}
ON	NOT
OFF	NOT
MAINT	FIRST
MAINT	SUBSEQUENT

The second set of hidden variables, Z_{it}, are boolean variables having domains {NONE, FULL}, which indicate whether unit i is producing power output during week t. The obvious binary constraints are defined between each X_{it} and the corresponding Z_{it}.

Constraint (9.1) – weekly power demand

Each demand constraint involves the U visible variables that relate to a particular week. The basic idea is to enforce a U-ary constraint between these variables which guarantees that enough of the variables will be ON to meet the power demand for the week. This constraint can be implemented as a table of either compatible or incompatible combinations, or as a procedure which takes as input the U variables and returns TRUE or FALSE. Our implementation uses a table of incompatible combinations. For example, suppose there are four generating units, with output capacities k_1=100, k_2=200, k_3=300, k_4=400. For week 5, the demand D_5=800. The following 4-ary constraint among variables $(Z_{1,5}, Z_{2,5}, Z_{3,5}, Z_{4,5})$ is created (incompatible tuples are listed).

$Z_{1,5}$	$Z_{2,5}$	$Z_{3,5}$	$Z_{4,5}$	comment (output level)
NONE	NONE	NONE	NONE	0
NONE	NONE	NONE	FULL	400
NONE	NONE	FULL	NONE	300
NONE	NONE	FULL	FULL	700
NONE	FULL	NONE	NONE	200
NONE	FULL	NONE	FULL	600
NONE	FULL	FULL	NONE	500
FULL	NONE	NONE	NONE	100
FULL	NONE	NONE	FULL	500
FULL	NONE	FULL	NONE	400
FULL	FULL	NONE	NONE	300
FULL	FULL	NONE	FULL	700
FULL	FULL	FULL	NONE	600

Because the domain size of the Z variables is 2, a U-ary constraint can have as many as $2^U - 1$ tuples. If this constraint were imposed on the X variables directly, which have domains of size 3, there would be 79 tuples ($3^4 - 5$) instead of 13 ($2^4 - 3$). This is one reason for creating the hidden Z variables: to reduce the size of the demand constraint.

Constraints (9.2) and (9.3) – earliest and latest maintenance start date

These constraints are easily implemented by removing the value FIRST from the domains of the appropriate Y variables.

Constraint (9.4) – continuous maintenance period

To encode this domain constraint in our formalism, we enforce three conditions using binary relational constraints over the Y's:

1. There is only one first week of maintenance.
2. Week 1 cannot be a subsequent week of maintenance.
3. Every subsequent week of maintenance must be preceded by a first week of maintenance or a subsequent week of maintenance.

Each of these conditions can be enforced by unary or binary constraints on the Y variables.

Constraint (9.5) – length of maintenance period

A maintenance period of the correct length cannot be too short or too long. For each unit i, each time period t, and every $t_1, t < t_1 < t + d_i$, the following binary constraint prevent a short maintenance period (disallowed tuple listed):

Y_{it}	Y_{it_1}
FIRST	NOT

To ensure that too many weeks of maintenance are not scheduled, it is only necessary to prohibit a subsequent maintenance week in the first week that maintenance should have ended. This results in the following constraint for each i and t, letting $t_1 = t + d_i$ (disallowed tuple listed):

Y_{it}	Y_{it_1}
FIRST	SUBSEQUENT

Constraint (9.6) – existence of maintenance period

This requirement is enforced by a high arity constraint among the Y variables for each unit. Only the weeks between the earlist start week and the latest start week need be involved. At least one $Y_{it}, e_i \leq t \leq l_i$, must have the value START. It is simpler to prevent them from all having the value NOT, and let constraints (9.4) and (9.5) ensure that a proper maintenance period is established. Thus the $(l_i - e_i + 1)$-arity constraint for each unit i is (disallowed tuple listed):

Y_{il_i}	...	Y_{ie_i}
NOT	NOT	NOT

Constraint (9.7) – no more than M units maintained at once

If M units are scheduled for maintenance in a particular week, constraints must prevent the scheduling of an additional unit for maintenance during that week. Thus the CSP must have $(M+1)$-ary constraints among the X variables which prevent any $M+1$ from having the value of MAINT in any given week. There will be $\binom{U}{M+1}$ of these constraints for each of the W weeks. They will have the form (disallowed tuple listed):

$X_{i_1 t}$	...	$X_{i_M t}$
MAINT	MAINT	MAINT

We see that this requires an exponential number in M of no-goods. If M is big, it may be beneficial to leave this constraint in a procedural (rather than relational) form.

Constraint (9.8) – incompatible pairs of units

The requirement that certain units not be scheduled for overlapping maintenance is easily encoded in binary constraints. For every week t, and for every pair of units $(i_1, i_2) \in N$, the following binary constraint is created (incompatible pair listed):

$X_{i_1 t}$	$Y_{i_2 t}$
MAINT	MAINT

Objective function (9.9) – minimize cost

To achieve optimization within the context of our constraint framework, we create a constraint that specifies that the total cost must be less than or equal to a set amount. In order to reduce the arity of the cost constraint, we introduce a simplification to the problem: we optimize cost by week instead of over the entire planning period. Therefore, the algorithm achieves an optimal solution to a more restricted cost function which may not optimize the original one.

We implemented the cost constraint as a procedure in our CSP solving program. This procedure is called after each X type variable is instantiated. The input to the procedure is the week, t, of the variable, and the procedure returns TRUE if the total cost corresponding to week t variables assigned ON or MAINT is less than or equal to C_t, a new problem parameter (not referenced in Fig. 2) which specifies the maximum cost allowed in period t. This is the only constraint in our formulation that is implemented procedurally.

4 Solution Procedure

One of our goals was to investigate the efficacy of various constraint satisfaction algorithms on maintenance scheduling CSPs. We first describe several algorithms which have been reported earlier, and then describe a new technique which takes advantage of the structure of optimization problems in the CSP framework.

4.1 Algorithms for CSPs

All of the algorithms we used incorporated a dynamic variable ordering heuristic, dubbed DVO. The first step is to process the domains of variables which have not yet been assigned values, marking as unavailable those values which are incompatible with the current partial assignment. If the entire domain of a variable becomes unavailable, then the algorithm backtracks, as the current partial assignment cannot lead to a solution. Otherwise, the variable with the fewest remaining compatible values is made next in the ordering. Ties are broken randomly. The variable ordering can differ on different branches of the search tree. Unless specified otherwise (under the LVO heuristic), one of the remaining compatible values is chosen arbitrarily.

The simplest algorithm we used is backtracking with dynamic variable ordering, or BT+DVO. Upon encountering a dead-end, BT+DVO tries to find a new value for the variable immediately preceding the dead-end variable. The BT+DVO algorithm proved to be ineffective on the maintenance scheduling CSPs, and was not used in the experiments reported below.

Another algorithm which does more work at each instantiation, by integrating an AC-3 based arc-consistency procedure [14] is BT+DVO+IAC. With IAC, values for future variables are removed not only if they are inconsistent with the current partial assignment, but also if they are not compatible with at least one value in the remaining domain of each other future variable. At the cost of more processing per node, BT+DVO+IAC increases the likelihood of detecting early that a partial assignment cannot lead to a solution.

The other algorithms are based on backjumping. This algorithm is a variant of backtracking, but after a dead-end can return to an earlier variable than the immediatedly previous one. The version of backjumping we use is called conflict-directed backjumping [16], and backjumping with dynamic variable ordering is called BJ+DVO. Look-ahead value ordering is a heuristic for ordering the compatible values in the domain of the current variable; it can be combined with BJ+DVO to yield BJ+DVO+LVO.

Learning in CSPs, also known as constraint recording, involves a during-search transformation of the problem representation into one that may be search more effectively. This is done by enriching the problem description by new constraints, also called *no-goods*, which do not change the set of solutions, but make the problem more explicit. The new constraints are essentially uncovered by resolution during the search process [15]. Learning comes into play at dead-ends; whenever a dead-end is reached a constraint explicated by the dead-end is recorded. Learning during search has the potential for reducing the size of the search space, since additional constraints may cause unfruitful branches of the search to be cut off at an earlier point. The cost of learning is that the computational effort spent recording and then consulting the additional constraints may overwhelm the savings. The kind of learning employed here takes advantage of processing already performed by the backjumping algorithm to identify the new constraint to be learned. It can be combined with backjumping and dynamic variable ordering and is called BJ+DVO+LRN. A high-level sketch of this algorithm is given in Fig. 4. When only constraints with i or fewer variables are recorded by learning, the result is called ith-order learning. We can also add LVO to produce BJ+DVO+LRN+LVO.

For more details about these and other constraint processing algorithms, see [11, 9, 8, 10, 7, 3, 16, 4].

4.2 Optimizing with CSPs

The constraint satisfaction framework is a decision procedure; any solution that can be found is equally good. We can find an optimal schedule by treating the maintenance scheduling problem as a series of CSPs. The procedure is described in Fig. 3. Initially, a schedule is found with a very high cost-bound. For the maintenance scheduling problems, this is C_t, the maximum cost per week. The cost-bound is then gradually lowered, with a new schedule found each time. Eventually, the cost-bound is so low that no schedule exists which meets it, and the last schedule found is optimal, within the limit of the amount by which the cost-bound was lowered. A more sophisticated control algorithm, based on a binary search approach, can be envisioned. In the experiments reported below, we used the simple decrement only technique. Another enhancement would be to permit different cost-bounds for different weeks.

Solution Procedure for Optimization
Input: A MSCSP with hard constraints, and an objective function.
Output: The lowest cost-bound for which a solution was found, and a solution with that cost-bound.

1. Set the cost-bound to a high value.
2. Until no solution can be found,
   (a) Add a constraint (or set of constraints) to the MSCP specifying that the value of the objective function is less than the cost-bound.
   (b) Solve the MSCSP using a constraint algorithm.
   (c) Decrement the cost-bound.
3. Return the last solution found, and the corresponding cost-bound.

Figure 3: The solution procedure for optimization.

4.3 Optimization with Learning

To make the optimization process more efficient, we introduce the notion of a memory that exists between successive iterations of step 2 in Fig. 3 The idea is to use a learning algorithm, such as BJ+DVO+LRN, to solve the maintenance scheduling CSPs (MSCSPs), and the new constraints introduced by learning are retained for use in later iterations. We call this approach *iterative learning.*

Retaining a memory of constraints is safe because as the cost-bound is lower the constraints become tighter. Any solution to an MSCSP with a certain cost-bound is also a valid solution to the same problem with a higher cost-bound. If the the cost-bound were both lowered and raised, as suggested in the previous section with a binary search approach, then some learned constraints would have to be "forgotten" when the cost-bound was raised.

5 Problem Instance Generator

One of our goals is to be able to determine the efficacy of various CSP algorithms and heuristics when applied to Maintenance Scheduling CSPs. To perform an experimental average-case analysis, we need a source of many MSCSPs. We have therefore developed an MSCSP generator, which can create any number of problems that adhere to a set of input parameters.

The input to the generator is a file containing most of the basic parameters, either explicitly enumerated or a kernel for generating all the necessary parameters by interpolation or by some parameterized distribution. The generator generates as many

Backjumping with learning

1. If all variables have been assigned values, then return this solution. Otherwise, select a variable using the dynamic variable ordering heuristic.
2. Find a compatible value for the current variable. If successful, go to 1. Otherwise, go to 3.
3. (Dead-end.) Find a subset of the variables with values assigned that are responsible for the dead-end. Add a new constraint which prohibits that combination from reoccuring. Select the latest variable in that subset to be the current variable, and go to 2.

Figure 4: Sketch of the BJ+DVO+LRN algorithm.

problem instances as necessary using the input parameters and then the problem instance is solved by the various algorithms. The maintgen program generator reads in a file and creates one or more MSCSP instances which can be solved by the CSP solver. The maintgen program uses a random number generator seed and a number indicating how many individual problems should be generated. The parameters given to the generator specify the fundamental size parameters: the number of weeks W, the number of generating units U, and the number of units which can be maintained at one time M. Also, the demand for some number of weeks is specified. The demand for weeks that are not explicitly specified is computed by a linear interpolation between the surrounding specified weeks. The initial maximum cost per week, and the amount it is to be decremented after each successful search for a schedule, are also specified.

The characteristics of the units, that is, their output capacities and required maintenance times, are not specified individually. Instead, these values are randomly selected from normal distributions whose means and standard deviations are specified. Maintenance costs are specified by the standard deviation and by a sample of weekly demands per unit. As with demand, values for weeks that are not given explicitly are interpolated. Operating costs are defined with exactly the same structure. The last piece of information is the number of incompatible pairs of units. The requested number of pairs is created randomly from a uniform distribution of the units.

Here is an example of an input file to the generator followed by a specification of the problem instance that was generated.

```
# lines beginning with # are comments
# first line has weeks, units, maximum simultaneous units
4 6 2
```

```
#
# next few lines have several points on the demand curve,
# given as week and demand.  Other weeks are interpolated.
0   700
3  1000
# end this list with EOL
EOL
#
# next line has initial max cost per week, and decrement amount
60000 3000
#
# next line has average unit capacity and standard deviation
200 25
#
# next line has average unit maintenance time and std. dev.
2 1
#
# next line has standard deviation for maintenance costs
1000
#
# next few lines have some points on the maintenance cost curve,
# first number is week, then one column per unit
0 10000 10000 10000 10000 10000 10000
3 13000 16000 19000 10000  7000 10000
#
# next number is standard deviation for operating costs
2000
# next few lines have some points on the operating cost curve,
# first number is week, then one column per unit
0 5000 5000 5000 5000 5000 5000
# the next line specifies the number of incompatible pairs
2
# and that's it!
```

Below is a corresponding generated problem instance.

```
# comments begin with #
# first line has weeks W, units U, max-simultaneous M
4 6 2
# demand, one line per week
700
800
900
```

```
1000
# next few lines has maximum cost per week.
# Cost must be <= max.
60000 3000
# one line per unit:
# capacity  maint length  earliest maint start  latest maint start date
194 1 0 3
171 3 0 3
209 1 0 3
166 1 0 3
219 2 0 3
217 2 0 3
# maintenance costs, one line per week, one column per unit
11085 10034 9374 8945 10858 10045
11056 11988 13670 10465 9301 10625
12745 14625 15422 10422 8099 7629
12534 15394 21098 9841 6748 9364
# operating costs, one line per week, one column per unit
4284 6857 3847 5050 5145 4998
5987 7352 1967 4635 6152 4635
3746 6475 5151 3988 8172 4131
6152 3436 5475 5600 4366 6070
# incompatible pairs of units (numbering starts from 0)
1 3
2 3
EOL
# and that's it!
```

The output problem instance is in a format which is recognized by our CSP solver.

6 Experimental Results

We present the results of experiments with two sets of 100 MSCSPs each. The smaller problems had 15 units and 13 time periods, resulting in 585 variables. The larger problems had 20 units and 20 time periods, resulting in 1200 variables. We conducted two experiments with each set of 100 problems. In the first we used the algorithms BJ+DVO and iterative learning based on BJ+DVO+LRN to solve the optimization task. In the second we compared the performance of all five algorithms described in section 4, using a fixed cost-bound that is close to the lowest feasible one.

6.1 Optimization with learning

In the first experiment, we tried to find an optimal schedule for each MSCSP in the smaller and larger sets, using BJ+DVO and iterative learning. Iterative learning used 6th-order BJ+DVO+LRN. The results are shown in Fig. 5 and Fig. 6.

For the 100 smaller problems, the cost-bound was set initially at 110,000 per week, and then reduced by 5,000 for each iteration. All 100 MSCSPs had schedules at cost-bound 85,000 and above. Only 38 had schedules within the 80,000 bound; at 75,000 only four problems were solvable. On the set of 100 larger MSCSPs, the cost-bound started at 150,000 per week and was reduced by 5,000. Schedules were found for all instances at cost-bound 120,000 and above. 97 instances had schedules at cost- bound 115,000 and 110,000; 11 at cost-bound 105,000; and two at cost-bound 100,000 and 95,000.

Iterative learning performed better, on average, than BJ+DVO on these random maintenance scheduling problems. For instance, on the set of smaller problems, after finding a schedule with cost-bound 95,000 the average number of learned constraints was 214. Tightening the cost-bound to 90,000 resulted in over twice as much CPU time needed for BJ+DVO (54.01 CPU seconds compared to 23.28), but only a 71% increase for iterative learning (29.41 compared to 17.20). Iterative learning was less effective on the larger MSCSPs. Although it required less CPU time on average, the improvement over BJ+DVO was much less than on the smaller problems.

6.2 Comparison of Constraint Algorithms

The second experiment utilized the same sets of 100 smaller MSCSP instances and 100 larger instances, but we did not try to find an optimal schedule. For the smaller problems we set the cost-bound at 85,000 and for the larger problems we set the cost-bound at 120,000. Each bound was the lowest level at which schedules could be found for all problems. We used the five algorithms described earlier to find a schedule for each problem. The results are summarized in Table 1.

Among the five algorithms, BJ+DVO performed least well on the smaller problems and best on the larger problems, when average CPU time is the criterion. BT+DVO+IAC was the best performer on the smaller problems and the worst on the larger problems. This reversal in effectiveness may be related to the increased size of the higher arity constraints on the larger problems. The high arity constraints, such as those pertaining to the cost-bound, the weekly power demand, and the existence of a maintenance period, become looser as the number of units and number of weeks increase. Earlier results [7] have indicated that more look-ahead is effective on problems with tight constraints, and detrimental on problems with loose constraints. Nevertheless backjumping remains an effective technique on the larger problems. Further experiments are required to determine how the relative efficacy of different algorithms is influenced by factors such as the size of the problem (number of weeks and units) and characteristics such as the homogeneity of the units.

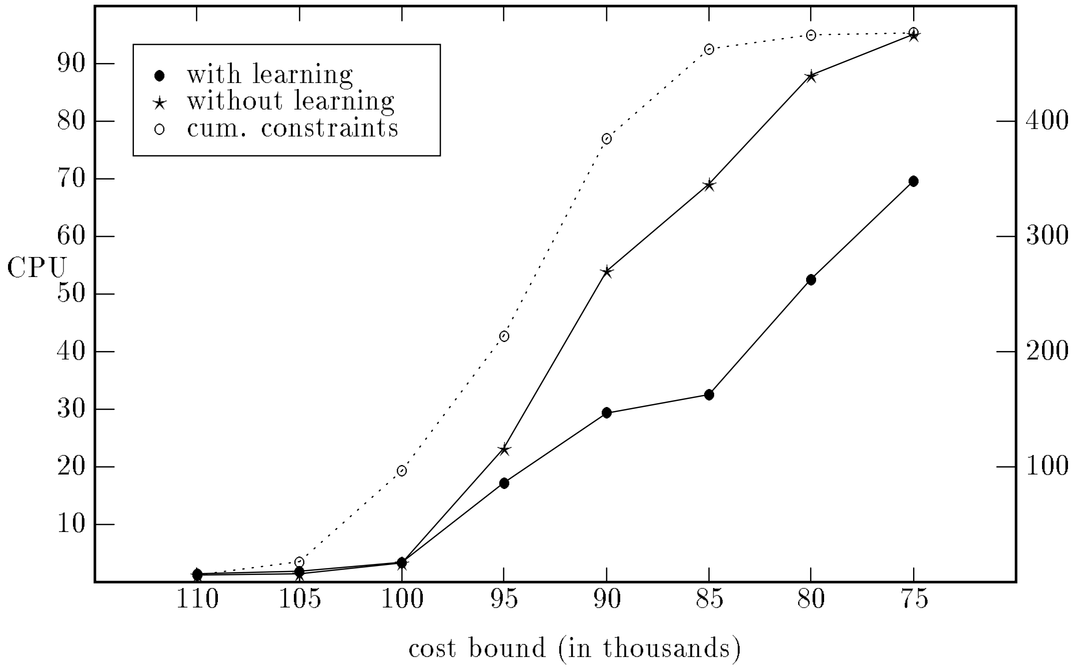

Figure 5: Average CPU seconds on 100 small problems (15 units, 13 weeks) to find a schedule meeting the cost-bound on the y-axis, using BJ+DVO with learning ($\bullet$) and without learning ($\star$). Cumulative number of constraints learned corresponds to right-hand scale.

7 Conclusions

The constraint satisfaction problems derived from the maintenance scheduling needs of the electric power industry are an interesting testbed for CSP algorithms. The problems have a mixture of tight binary constraints, such as those that bind the X and Y variables together, and loose high arity constraints, such as those that ensure that at least one maintenance period is scheduled for each unit. The most promising algorithm for these problems is iterative learning. Further studies on larger maintenance scheduling CSPs is required to determine whether one algorithm dominates the others as problem size increases.

A challenging problem that is difficult to formalize is to find the best way to encode the requirements of a problem such as maintenance scheduling into constraints of a CSP. In section 3 we discussed some of the trade-offs involved in, for example, adding "hidden" variables in return for a smaller number of tuples in high arity constraints. This is an important area for future research that has the potential of greatly impacting the applicability of the constraint satisfaction framework to problems from science and industry.

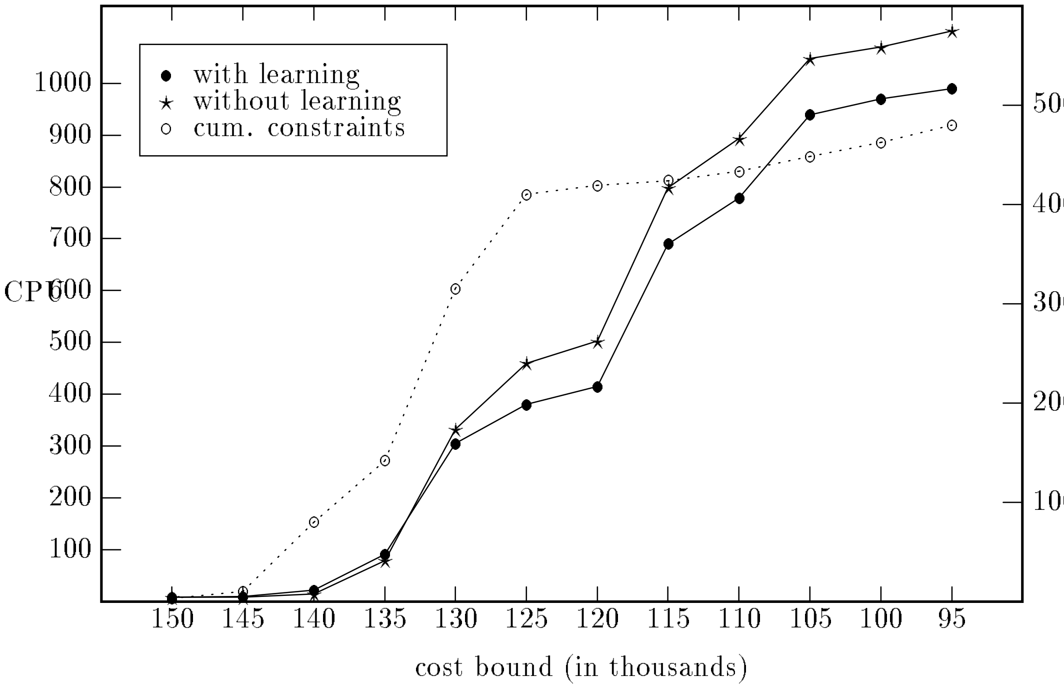

Figure 6: Average CPU seconds on 100 large problems (20 units, 20 weeks) to find a schedule meeting the cost-bound on the y-axis, using BJ+DVO with learning ($\bullet$) and without learning ($\star$). Cumulative number of constraints learned corresponds to right-hand scale.

References

[1] T. M. Al-Khamis, S. Vemuri, L. Lemonidis, and J. Yellen. Unit maintenance scheduling with fuel constraints. *IEEE Trans. on Power Systems*, 7(2):933–939, 1992.

[2] Roberto Bayardo and Daniel Mirankar. A complexity analysis of space-bounded learning algorithms for the constraint satisfaction problem. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 298–304, 1996.

[3] Rina Dechter. Enhancement Schemes for Constraint Processing: Backjumping, Learning, and Cutset Decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[4] Rina Dechter. Constraint networks. In *Encyclopedia of Artificial Intelligence*, pages 276–285. John Wiley & Sons, 2nd edition, 1992.

[5] J. F. Dopazo and H. M. Merrill. Optimal Generator Maintenance Scheduling using Integer Programming. *IEEE Trans. on Power Apparatus and Systems*, PAS-94(5):1537–1545, 1975.

Algorithm	Average CC	Nodes	CPU
100 smaller problems:			
BT+DVO+IAC	315,988	3,761	51.65
BJ+DVO	619,122	8,981	70.07
BJ+DVO+LVO	384,263	5,219	54.48
BJ+DVO+LRN	671,756	8,078	67.51
BJ+DVO+LRN+LVO	476,901	5,085	57.45
100 larger problems:			
BT+DVO+IAC	7,673,173	32,105	694.02
BJ+DVO	2,619,766	28,540	460.42
BJ+DVO+LVO	6,987,091	26,650	469.65
BJ+DVO+LRN	5,892,065	27,342	521.89
BJ+DVO+LRN+LVO	6,811,663	26,402	475.12

Table 1: Statistics of five algorithms on MSCSPs.

[6] G. T. Egan. An Experimental Method of Determination of Optimal Maintenance Schedules in Power Systems Using the Branch-and-Bound Technique. *IEEE Trans. SMC*, SMC-6(8), 1976.

[7] Daniel Frost. *Algorithms and Heuristics for Constraint Satisfaction Problems.* PhD thesis, University of California, Irvine, CA 92697-3425, 1997.

[8] Daniel Frost and Rina Dechter. Dead-end driven learning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 294–300, 1994.

[9] Daniel Frost and Rina Dechter. In search of the best constraint satisfaction search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 301–306, 1994.

[10] Daniel Frost and Rina Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 572–578, 1995.

[11] R. M. Haralick and G. L. Elliott. Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, 14:263–313, 1980.

[12] Henry Kautz and Bart Selman. Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 1194–1201, 1996.

[13] Hyunchul Kin, Yasuhiro Hayashi, and Koichi Nara. An Algorithm for Thermal Unit Maintenance Scheduling Through Combined Use of GA SA and TS. *IEEE Trans. on Power Systems*, 12(1):329–335, 1996.

[14] A. K. Mackworth and E. C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[15] Alan K. Mackworth. The logic of constraint satisfaction. *Artificial Intelligence*, 58:3–20, 1992.

[16] Patrick Prosser. Hybrid Algorithms for the Constraint Satisfaction Problem. *Computational Intelligence*, 9(3):268–299, 1993.

[17] J. Yellen, T. M. Al-Khamis, S. Vemuri, and L. Lemonidis. A decomposition approach to unit maintenance scheduling. *IEEE Trans. on Power Systems*, 7(2):726–731, 1992.

[18] H. H. Zurm and V. H. Quintana. Generator Maintenance Scheduling Via Successive Approximation Dynamic Programming. *IEEE Trans. on Power Apparatus and Systems*, PAS-94(2), 1975.