Solving Partial Constraint Satisfaction Problems using Local Search and Abstraction

Qiang Yang and Philip W. L. Fong*

Department of Computer Science
University of Waterloo

Abstract

Partial constraint satisfaction problems (PCSPs) were proposed by Freuder and Wallace to address some of the representational difficulties with traditional constraint satisfaction techniques. However, the reasoning method of their proposal was limited to traditional backtracking based algorithms. In this paper, we extend the PCSP model by associating it with a local search algorithm, which has found great successes in solving many large scale problems in the past. Furthermore, we extend the combined model to incorporate abstract problem solving, and show that the extended model has not only the advantages of both PCSP and local search, but also a number of new features useful for scheduling applications. We demonstrate the feasibility of our approach by an application to a university course scheduling domain.

Keywords: Artificial Intelligence, Expert Systems, Constraint Satisfaction, Knowledge-Based Scheduling, Heuristic Search.

*The authors are supported in part by an operating grant from the Natural Sciences and Engineering Research Council of Canada, and a grant from the Information Technology Research Centre of Ontario.

1 Introduction

Constraint Satisfaction Problems (or CSPs) have found many applications ranging from temporal reasoning[1], machine vision[14], to scheduling problems[3, 4]. A CSP can be formulated abstractly as consisting of a set of variables, each variable is associated with a domain of values that can be assigned to the variable[9, 10]. In addition, a set of constraints exists that defines the permissible subsets of assignments to variables. The goal is to find one (or all) assignment of values to the variables such that no constraint is violated.

There are two major problems with the traditional CSP formulation, and reasoning methods for solving a CSP. First, the traditional definition of CSP has met with some *representational* difficulties in practice, due to its overly strict modeling. In particular, the classical formulation of CSPs requires that *all constraints* must be satisfied. As such, it can be overconstrained by the constraint set and in many cases, admits no solution. In practice, however, it is sometimes the case that certain constraints can be violated occasionally, or weakened to some degree. In addition, for domains where there is a fixed bound on computational resources, it may not be possible to find a complete solution for a given CSP. Instead, a partial solution, where some constraints are violated, is usually good enough for many purposes. To address this problem, Freuder and Wallace introduced the concept of Partial Constraint Satisfaction Problems [5, 6], also called PCSPs. Such a new formulation is used to capture the idea that certain constraints can be relaxed, or violated in a solution.

Another problem of traditional CSPs is the inefficiency of their associated reasoning algorithms. Most CSP algorithms proposed in AI area are backtracking-based, which systematically assign a value to each variable. When an inconsistent assignment is encountered, the algorithm backtracks to another assignment. However, empirical results have shown that such backtracking based methods are capable of solving only small scale CSPs, and cannot meet the practical needs for solving problems with large sets of constraints or variables. An alternative method, based on local search, was recently proposed by several authors, which has been shown to be promising for solving very large scale constraint satisfaction problems. The method is based on a heuristic known as *minimal conflict*, which when used with a local, hill-climbing search algorithm performs assignment in a random fashion. Several authors have demonstrated its efficiency on the N-queens problem, where N is on the order of several million[13, 11].

As stated above, although the PCSP model extends the representation aspect of CSPs, the reasoning technique proposed by Freuder and Wallace is still based on traditional backtracking methods. Therefore, it will suffer from the same problem of inefficiency as does traditional CSPs. On the other hand, the local search method addresses the efficiency issue of reasoning about a CSP, but the representational model was still aimed at solving a CSP exactly. An extension would therefore be to merge the representational novelty of PCSPs with the reasoning superiority of the local search method.

In this paper, we combine the two novel techniques, PCSP and local search, for formulating and solving constraint satisfaction problems. We will show that such a combination not only keeps the advantage of both techniques, but has some additional properties. In partic-

ular, the combined representation allows for a unified way to model and reason about different types of constraints, including hard constraints, soft constraints, and meta-constraints. In addition, with the new framework the problem of *revising* past solutions can be easily addressed. Furthermore, where constraints can be partitioned according to their relative importance, an abstraction based method can be naturally applied as well.

The combined framework for solving CSPs has been implemented in LISP and applied to a course scheduling domain. The implemented system, WATCOURSE effectively demonstrates the feasibility of our approach.

Below, we first review the PCSP representation and the local search method. Then we discuss how the two can be combined into a unified system. We then demonstrate the feasibility of our approach through a course scheduling domain.

2 The PCSP Model

A CSP consists of a set of variables, a domain for each variable, and a set of constraints. Formally, a CSP consists of the following components:

1. V is a set of variables,
2. D is a set of domains, i.e., sets containing values to be assigned to the variables.
3. C is a set of constraints.

An often used example for demonstrating the CSP is the N-queens problem, where N queens are to be placed on an $N \times N$ board. The constraints are that no two queens can be on the same row, column, or diagonal. One way to look at the domain is to place a queen on each row. If we take each row X as a variable, then a column number v is a domain value for X. Represented in this way, the constraints are simply that no two queens can be on the same column, and that no two queens can be on the same diagonal.

As an example, the 3-queens problem can be modeled as:

1. $V = \{X_1, X_2, X_3\}$, representing the three rows of the problem.
2. $D = \{D_1, D_2, D_3\}$, where $D_i = \{1, 2, 3\}$.
3. C is a predicate, such that

$$C((X_i, v_1), (X_j, v_2)) = ture, \text{ iff } |v_1 \neq v_2| \text{ and } |j - i| \neq |v_2 - v_1|.$$

The third constraint says that no two queens can attack each other.

A CSP may not have a solution due to its constraints. For example, the 3-queens problem shown in Figure 1 has no solution. A partial constraint satisfaction problem, or a PCSP, is a relaxation of the original CSP. For a given CSP, one might relax it by enlarging the domain of a constraint, removing a variable or a constraint, or enlarging the domain of a variable. Any

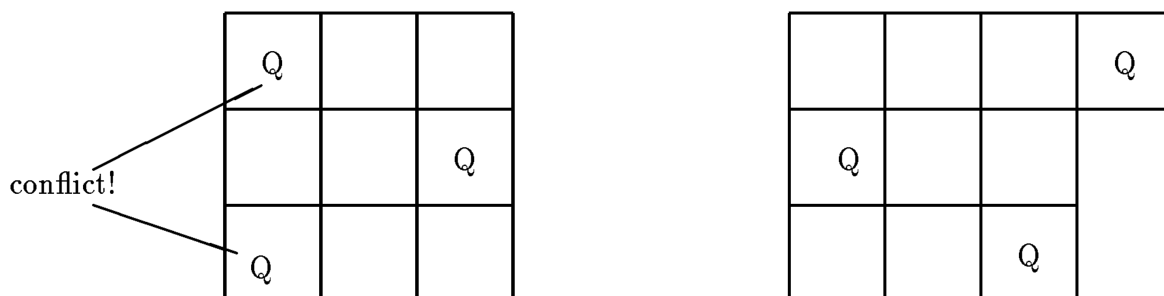

Figure 1: An unsolvable CSP problem and its relaxation.

of these operations gives an extended CSP. For example, in the above 3-queens example, if one enlarges the domain of the first variable X_1 by including a new value 4, then the problem becomes easy to solve (See Figure 1). The new CSP is therefore a relaxed version of the original CSP.

Formally, a PCSP is a partially ordered set of CSPs, with a common root. The root is the original CSP. The rest of the nodes in the graph are CSPs obtained form the original one through a sequence of relaxation operations. Consult Figure 2 for an intuitive explanation.

Given two CSPs in the graph, one can measure the distance between them, by associating a PCSP with a metric. The metric might measure the difference in the number of solutions, the number of added domain values, or it might measure the number of missing (or relaxed) constraints. Solving a PCSP then becomes a problem of finding a solution to a relaxed CSP in the space of PCSP, so that the distance metric between the solution and the optimal solution is within a certain bound, according to the metric. To ensure that the space of partial CSPs is restrained, two special bounds are useful. The first one is a sufficient bound, which specifies that a solution to a relaxed CSP is good enough, if the metric distance between the solution and the optimal solution is within this bound. The second one is a necessary bound, which specifies that the space of CSPs under consideration must all contain solutions that are within the bound. This effectively restricts the size of the problem space under consideration. For example, for the 3-queens problem, since we know that the 4-queens problem has a solution, the necessary bound Nec can be set to 7, since 7 squares are added to convert a 3-queens to a 4-queens problem. Similarly, one might set Suff to be 2, which states that it is permitable to find a solution where two values are added to some variables' domains. Note that by setting Suff to zero corresponds to the original CSP.

To find a solution, Freuder and Wallace proposed a series of backtracking based methods. Such methods have been characterized as "constructive" algorithms by Minton *et al.*[11], since they all start solving a CSP from scratch. At each step, a new variable is instantiated, or a CSP is relaxed. The advantage of constructive type of methods is that they are complete: if there is a solution, it will be found. The disadvantage of constructive methods is that, by assigning one variable value at a time, they are often too conservative, the result of which reduces their speed so dramatically that they are often not useful for practical purposes.

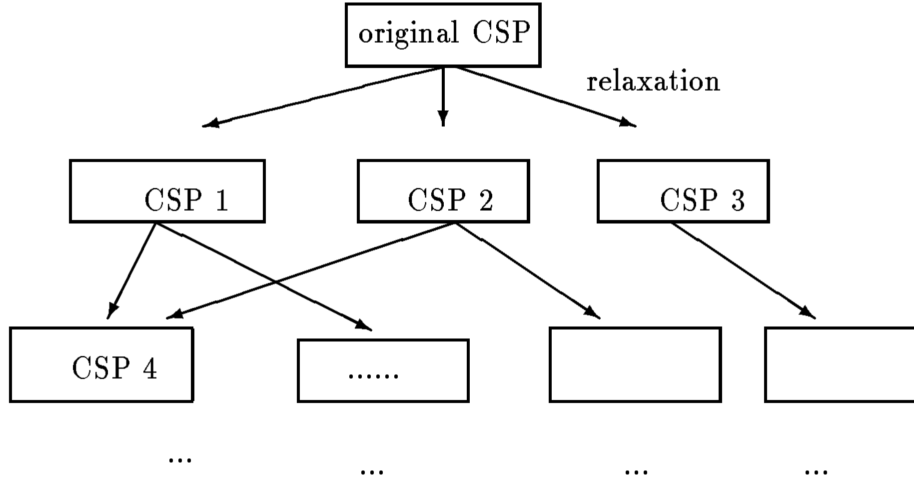

Figure 2: The problem space of partial constraint satisfaction.

3 Local Search with Min-Conflict Heuristic

An alternative algorithm was proposed to address the efficiency issue of solving CSPs. Such an algorithm, known as a *local search method*, performs a hill-climbing search. It usually starts with a complete assignment of values to variables, by certain random process. Then it *repairs* the initial assignment by changing the assignment of certain variables whose values violate some constraints. The process repeats until no more constraint violation remains.

The hill-climbing process of local search is guided by a heuristic known as a *min-conflict* heuristic. Stated simply, for a given set of variables with value assignments, the min-conflict heuristic finds a variable X whose value v is in conflict. Then it finds another value v' in the $Domain(X)$, such that the number of conflicts by v' is minimal among all values in X's domain. Ties are broken randomly. More formally, let $conf(X, v)$ be the number of conflicts as a result of assigning v to X. Then if $conf(X, v) \neq 0$, then

$$X := v', \text{ where } conf(X, v') = \min_{u \in Domain(X)} conf(X, u).$$

The local search method coupled with a min-conflict heuristic has many advantages over a backtracking method. First, because of its simplicity, it is easy to experiment with and implemented for different applications. Second, when the initial assignment is close to the final solution, the number of repairs needed to reach the final solution is relatively small, making the method extremely efficient. Third, in domains where revision of past schedule occurs often, and where the repair is limited locally, the local search method is very natural and efficient.

However, since local search is a greedy method, it is possible that it can be trapped into a local minimum. Experiments [13, 11] have shown that for many interesting domains, such worry is unnecessary. For example, implementations of the local search method by Sosic and

Gu [13] and Minton *et al.* [11] have been able to solve N-queens problems, where N is as high as several million, in less than one minute. Other supporting evidence comes from space telescope scheduling applications[11], and the large scale graph coloring and space shuttle payload scheduling problems [17].

4 Solving PCSPs with Local Search Methods

In the previous two sections, we reviewed a partial constraint satisfaction model, and a local search method for solving traditional constraint satisfaction problems. Partial constraint satisfaction problems was introduced by Freuder and Wallace to represent approximate solutions to CSP, but the reasoning methods that they introduced was based on traditional backtracking methods. In this section, we extend their PCSP model to include a local search reasoning procedure. We show that the combination of PCSP and local search has the advantages of both systems. That is, a system can reason about and search for an approximate solution to a CSP, and that such search is efficient. In addition, we show that the combined system exhibit additional representational properties.

Recall that PCSP is defined as a partially ordered set $\mathcal{P}$ of CSPs, with the common root node being the original CSP. Each of the rest of the CSPs is obtained by a sequence of "weakening" operations applied to the root. On the other hand, a local search algorithm works within a space $\mathcal{S}$ of complete assignments, starting from some initial assignment. Since each node in this space corresponds to an approximate solution to the original CSP, each node in R is also a solution to some weakened CSP in $\mathcal{P}$. Therefore, the two spaces are related to each other via the following relations:

1. $\forall s \in \mathcal{S}. \exists P \subseteq \mathcal{P}$, such that s is a solution of all CSPs in P. That is, every complete assignment corresponds to a subset of relaxed CSPS.

2. $\forall p \in \mathcal{P}. \exists S \subseteq \mathcal{S}$, such that S are all solutions for p. That is, every CSP corresponds to a subset of complete assignments.

3. $\forall s_1, s_2 \in \mathcal{S}$, a path exists from s_1 to s_2 in $\mathcal{S}$, if $\exists p_1, p_2 \in \mathcal{P}$ such that s_1 is a solution of p_1, s_2 is a solution of p_2 and there is a path from p_1 to p_2 in $\mathcal{P}$.

To find a solution to a PCSP using a constructive method, together with the necessary bound Nec and sufficient bound Suff, a top-down process is followed (See Figure 2). This solution process starts with a original CSP, and gradually weakens it until a CSP is found with a solution within the bounds Suff and Nec.

However, to search in the space of PCSP model, one does not have to start with the root node. Instead, one can also start from an internal node in Figure 2. This corresponds to using a local search for solving a PCSP. The solution process with the local search starts from a initial solution which may not be a strict solution to the original CSP. This initial assignment must satisfy the necessary bound Nec, but it may not satisfy the sufficient bound Suff. Thus, the whole local search process can be thought of as starting from a internal node

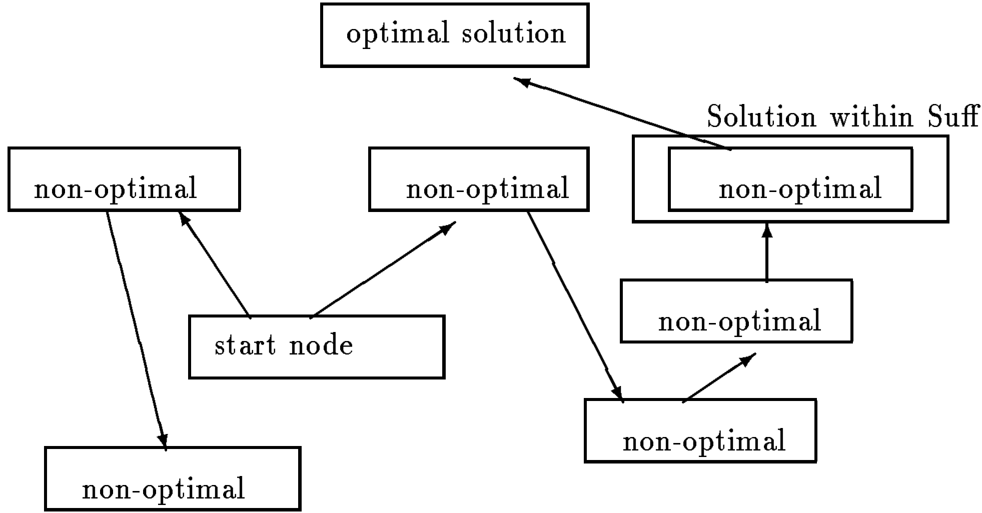

Figure 3: Local search in a space of solutions.

in $\mathcal{P}$, and following a path along which the total number of conflicts is decreasing. The process stops when a modified assignment is found which satisfies the sufficient bound Suff (See Figure 3). Note that the local search discussed here is a modified one; we no longer search for a precise solution to the original CSP, as done by Sosic and Gu, and Minton *et al.*, instead our local search is looking for a solution to a weakened CSP.

The algorithms that implement the above ideas are shown in Tables 1 and 2. Table 1 shows the initialization algorithm. It basically scans through the variable set, picking a value for each variable that minimizes the cost of the current assignment. If there are several values in a domain with minimal costs, then a choice is made randomly. Tables 2 takes the initialized solution from the initialization algorithm, and performs a repairing operation.

Clearly our algorithm has the advantage being able to model approximate solutions to a CSP, as originally proposed by the PCSP model, as well as the advantage of being efficient, as with the local search method using the min-conflict heuristic. Below, we consider three additional advantages of the method, namely, the ability to represent different types of constraints easily and using abstraction in problem solving, as well as the property of naturally representing and solving a solution revision problem.

5 Hard and Soft Constraints

The constraints dealt with by both Freuder and Wallace with their PCSP model, and Sosic and Gu, and Minton *et al.* with local search, are *hard* constraints. A hard constraint is one that *has to be* satisfied. As such, a hard constraint is usually a binary function, giving a value of either true or false. For example, in the N-queens problem, two queens either attack

Input: A set of variables V, Domains D, Constraints C, and a necessary bound Nec,
Output: sol, an initial assignment of domain values to variables.

Algorithm INIT
1. $sol := \emptyset$;
2. **for** every variable X in V, do
3. **if** v is a value in Domain(X), such that
   the cost of applying C to (X, v, sol) is minimal,
   then $sol :=$ append($\{(X, v)\}, sol$);
4. **endfor**
6. **if** Cost(sol) $<$ Nec **then return**(sol), **else** goto step 2.

Table 1: Initialization algorithm for local search.

Input: A set of variables V, Domains D, Constraints C, an initial solution sol, and a sufficient bound Suff. **Output:** sol, a final assignment of domain values to variables.

Algorithm Local-Search
1. **until** Cost(sol) $\leq Suff$ do
2. Let v be a value of an variable X, such that $(X, v) \in sol$, and Cost$(X, v) \neq 0$.
   If v' is another value, such that Cost$(X, v', sol - \{(X, v)\})$ is minimal,
   then $sol :=$ append($\{(X, v')\}, sol - \{(X, v)\}$);
4. **end until**
5. **return**(sol).

Table 2: Initialization algorithm for local search.

each other or they do not, there is no intermediate case. In contrast, a soft constraint is one that can be broken (or relaxed), and there is often a preference that it should be satisfied as much as possible. In addition, a soft constraint can have a variable degree of satisfaction[3]. For example, in the N-queens problem, one might specify that it is preferred that the first two queens do not attack each other, but if necessary, it is acceptable if the constraint is violated. One might also specify a preference that as many queens in a solution should be positioned to the left hand side as possible.

An orthogonal issue to hard and soft constraints concerns the importance of constraints. In particular, a soft constraint may not be a *less* important one, and conversely, a hard constraint may be of low importance. For example, in the N-queens problem it may be more important to place the first queen to close to the left column as much as possible. The distinction between importance of constraints and their hardness is particularly useful when a hard constraint is easy to satisfy, while a soft one is hard to satisfy. For example, in a course scheduling domain, it is often easy to satisfy the hard constraint that a course can be taught by one teacher only, but hard to satisfy preference constraints.

We model hard and soft constraints in a unified framework. Let C_i be a constraint.In our framework, C_i can be interpreted as a function, which takes as input a variable X, a value v, and an assignment A to the rest of the variables. It returns an natural number n. If $n = 0$ then the constraint is satisfied. Otherwise, n represents the *cost* of the assignment as a result of assigning v to X. Hence the cost value is a measure of how bad the assignment is due to $X := v$.

Given a list of constraints C_i represented as above, where $i = 1, 2, \ldots m$,we assume that the user has assigned to each constraint an importance value from 1 to k, with k the most important and 1 the least. The satisfaction of the set of all constraints can be collectively represented as a vector. The jth element of the vector is a sum of all constraints among C_i, $i = 1, 2, \ldots m$, that are of equal importance, j. This vector is called the *cost* of assigning v to X, given the current assignment A to the rest of the variables. Formally,

$$cost(X, v, A) = \langle E_1, E_2, \ldots, E_k \rangle$$

where $E_j = \Sigma\{C_i(X, v, A) \mid importance(C_i) = j\}$.

Finally, our modeling requires that the *cost* of a complete, approximate solution s is the vector sum over all variable assignments. Formally,

$$cost(s) = \Sigma\{cost(X, v, A) \mid for\ all\ (X, v) \in A\}$$

where the sum Σ is a vector summation.

To compare the cost of different assignments, we define a vector comparison operator as follows: a vector $V_1 = \langle A_1, A_2, \ldots, A_k \rangle$ is less than (denoted by $<$) $V_2 = \langle B_1, B_2, \ldots B_k \rangle$, if $\exists j \leq k$, such that $A_i = B_i$ for $i < j$, and $A_j < B_j$.

Using the above representation method, we can model the constraints in the N-queens problem as follows.

No two queens can be on the same diagonal.

$$C_d(X_i, v_1, A) = 1 \text{ if } \exists (X_j, v_2) \in A. |j - i| = |v_2 - v_1|. \text{ Else } 0.$$

No two queens can be on the same column.

$$C_n(X_i, v_1, A) = 1 \text{ if } \exists (X_j, v_2) \in A . v_1 = v_2. \text{ Else } 0.$$

Preferring left columns.

$$C_l(X_i, v_1, A) = v_1.$$

With the above definitions, it is now possible to incorporate all constraints, hard and soft, into a PCSP model with local search. Let $\emptyset$ denote a vector whose elements are all 0. During a local search process, let A be the current assignment. A search is made to look for a variable X with assignment v, such that $cost(X, v, A - \{(X, v)\}) \neq 0$. Then a another search is made in the domain of X, to look for a value v' with a minimal cost value. The minimality test utilizes the vector "$<$" operator. Finally, v' is assigned to X.

As in the local search model, sufficient and necessary bounds are given to control search. Given constraints of multiple levels of importance, both the sufficient and the necessary bounds are vectors. In addition, through settings of the elements in the sufficient bound vector, one can specify both hard and soft constraints. For example, suppose that there are three constraints C_1, C_2 and C_3, with C_1 the least important and C_3 the most important constraints. In addition, suppose that C_1 and C_3 are hard constraints, while C_2 is a soft constraint. Then the information can be represented by a sufficient bound $\langle 0, 10, 0 \rangle$, specifying a tolerance of value 10 for C_2. Thus, by setting a sufficient bound element to zero, one can represent a set of hard constraints. On the other hand, by setting an element to a non-zero value, one specifies a soft constraint.

6 Abstract Search

A second feature of our combined model is its ability to support *abstract* problem solving. The importance values assigned to constraints naturally define a hierarchy of problem spaces (See Figure 4). At the highest level (*i.e.* k^{th} level) are the constraints that are the most important. A solution to a PCSP can first be found in this space by considering only this set of most important constraints. Then the system *refines* the solution in successively more detailed spaces. During the refinement of an ith level solution at the $(i-1)^{th}$ level, the $(i-1)^{th}$ level constraints are appended at the end of the vector of constraints. The process continues until a time bound is exceeded, a sufficient bound is met, or a local minimum is encountered. If a local minimum is found without satisfying the sufficient bound, then the system can *backtrack* to the next higher level and try finding an alternative solution. Thus, the hierarchical system can be understood as a result of combining local search and backtracking problem solving. That is, in the vertical direction a backtracking method is used, while in the horizontal direction a local search method is used.

Abstract problem solving has received a lot of attention in AI planning area[12, 15, 16, 8]. There are two central issues about abstraction that are being addressed in planning: the question of how to effectively use an abstraction hierarchy, and the issue of how to find

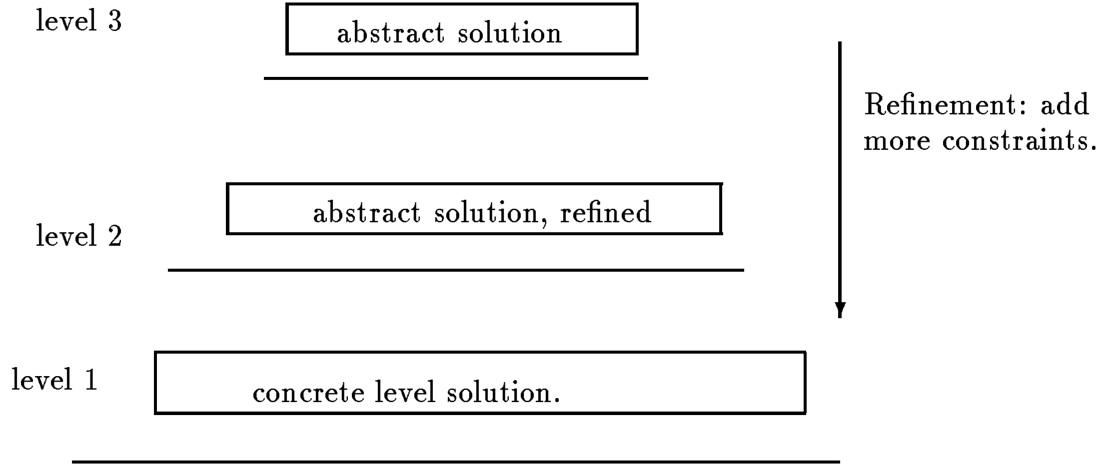

Figure 4: Refinement of an abstract solution.

a good abstraction hierarchy. In solving PCSPs, the same two issues are also of concern. Similarly, solutions proposed in planning area to address the above issues are also applicable to solving PCSPs. First, given an abstraction hierarchy, the refinement process should have the property that all abstract level achievements in satisfying the abstract constraints are preserved. That is, when a new value is assigned to a variable, care should be taken to ensure that few abstract constraints are violated. In planning, if a refinement does not violate any abstract level constraints, then it is called a *monotonic refinement*. With PCSP models, the definition for monotonic refinements can be stated as:

> If an abstract constraint C is violated N times in a solution, then during refinement of the solution, C cannot be violated more than N times either.

Figure 5 presents the flow chart of our implemention of the abstraction system. After the initial solution is found, the abstraction level counter i is first set to k. Then a loop repeats the following operations. First, an evaluation function is constructed for constraints on level i. Then a refinement of a previous solution is found at this level using the evaluation function. Finally, i is decremented by one. The loop repeats until the system is at the concrete level ($i = 0$), and the solution cost is within the sufficient bound, Suff. Note that the function "refine" is implemented simply as a call to the procedure "local-search" presented in Table 2.

The second issue regarding the use of abstraction is that of finding *good* abstraction hierarchies. A good abstraction hierarchy should be one that ensures improvement in efficiency, and furthermore, in the quality of solutions. In planning, several properties have been proposed to address this issue. An *ordered monotonic hierarchy* [8, 7] is one that leaves all abstract constraints intact when refining a solution at a lower level. Another property is known as the *downward refinement property*[2], whereby every abstract solution has at least one refinement. Experiments and theoretical analysis in planning has demonstrated that

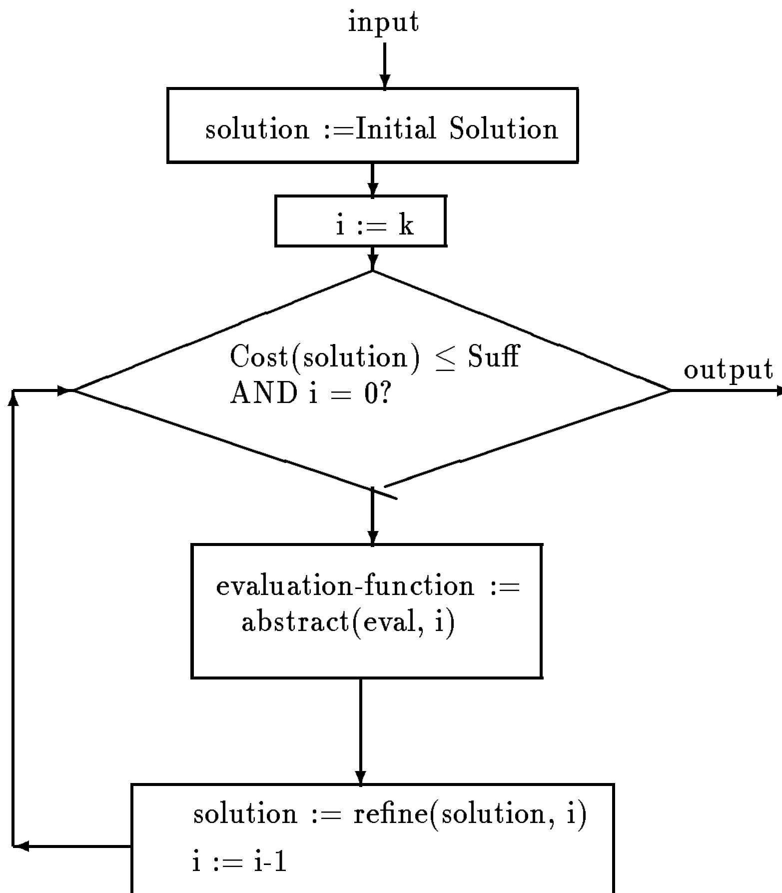

Figure 5: An overview of the hierarchical PCSP procedure.

they are effective in improving the search performance dramatically. For example, in the best case, one can reduce search time from exponential to linear in solution length. An advantage of these properties is that they enable one to automatically construct an abstraction hierarchy. Although exploring their duals in the context of PCSPs is beyond the scope of the current paper, we do intend to investigate this issue further in future research.

7 Revision

Another feature of of our combined model is its ability to revise solutions efficiently. Revision of a given solution may be necessary because of changes made in variable domains and constraints. Recall that the local search algorithm requires an initial assignment before a search is conducted. When revising an existing solution, the initial assignment is set to be the existing solution itself. Thus, the model naturally accommodates solution revision operations.

An important aspect of solution revision is to maintain the *stability* of the revision process. Typically, when a few variables in an existing solution are re-assigned, or when constraints are added, a rippling effect can occur. For example, when a variable X's value is changed, a constraint may force the value of another variable Y to be changed. This will in turn cause Z to be changed and so on. In many situations where stability of the organization is of concern, a long chain of changes is not desirable. Instead, one would like the rippling effect to die out when it reaches certain distance from the first change.

This *damping* effect on a chain of revision operations can be easily implemented via a soft constraint. Let s_0 be the initial solution to be revised and let s be a current solution. Then a stability constraint C is defined as

$$C = |(s_0 - s) \cup (s - s_0)|.$$

By assigning this constraint an appropriate importance value, it is possible to control the stability of the revision process by preferring to a change that is as close to the original solution as possible.

8 Experiments in the N-queens Domain

To test the effectiveness of our approach, we have conducted an experiment in the N-queens domain. The main purpose of the experiment is to verify our prediction that the hierarchical version of the PCSP model is more powerful than one without using abstraction. In particular, we have run two sets of experiments with the N-queens problem, one with abstraction and the other without. The one with abstract search has two levels of abstraction. On the top level are the constraints C_n and C_d (See Section 5), which state that no two queens can conflict with each other. At the bottom level we satisfy an additional constraint C_l, stating a preference for a leftmost column. In actual implementation, the top level search is guided by the constraint vector $\langle (C_n, C_d) \rangle$, and at the bottom level the vector $\langle (C_n, C_d), (C_l) \rangle$ is

Number of Queens	Abstraction	No Abstraction
10	(2 56)	(8 38)
20	(2 198)	(12 154)
30	(2 455)	(14 325)
40	(2 814)	(18 581)
50	(2 1262)	(24 870)
60	(4 1793)	(26 1303)
70	(4 2534)	(34 1700)
80	(4 3252)	(36 2283)
90	(10 4097)	(42 2814)

Table 3: Comparison of solution quality of search with and without abstraction. Time-bound is 17 seconds.

used. For the search without abstraction, the latter vector is the one used as constraints. Recall that by this vector the constraints C_n and C_d are considered more important than the constraint C_l.

Table 3 describes the comparisons in the *quality* of solutions found, taking both sets of constraints as soft ones, and giving both problem-solvers an equal time bound (18 CPU seconds on a Sun4 Sparc Station). Each item in the table is a vector (x, y), where x is the number of remaining violations with the most important constraint, and y the least important one. As can be observed from the table, the quality of solutions using abstraction is much better than without using abstraction, since with abstraction there are much less violations with the important constraints. However, it can also be observed that the quality of solutions found by the abstract PCSP model, in terms of the constraint C_l, is worse than that without abstraction. This is also expected, since a solution found at the abstract level places a strong constraint on search at the lower level. Thus, during the refinement of an abstract solution, it is more difficult to move away from a local minimum where most of the important constraints are satisfied. This observation reveals that there is in general a trade-off between the satisfaction of constraints of different degrees of importance, when different problem sovlers are used.

9 Application to Course Scheduling

We now turn our attention to a practical application of our framework, in a course scheduling domain. Course scheduling is an ideal domain for applying constraint satisfaction techniques, since it is full of different types of constraints. Some constraints are more important than others. Also, compromises in constraints are constantly made in this domain, making PCSP models more realistic to apply.

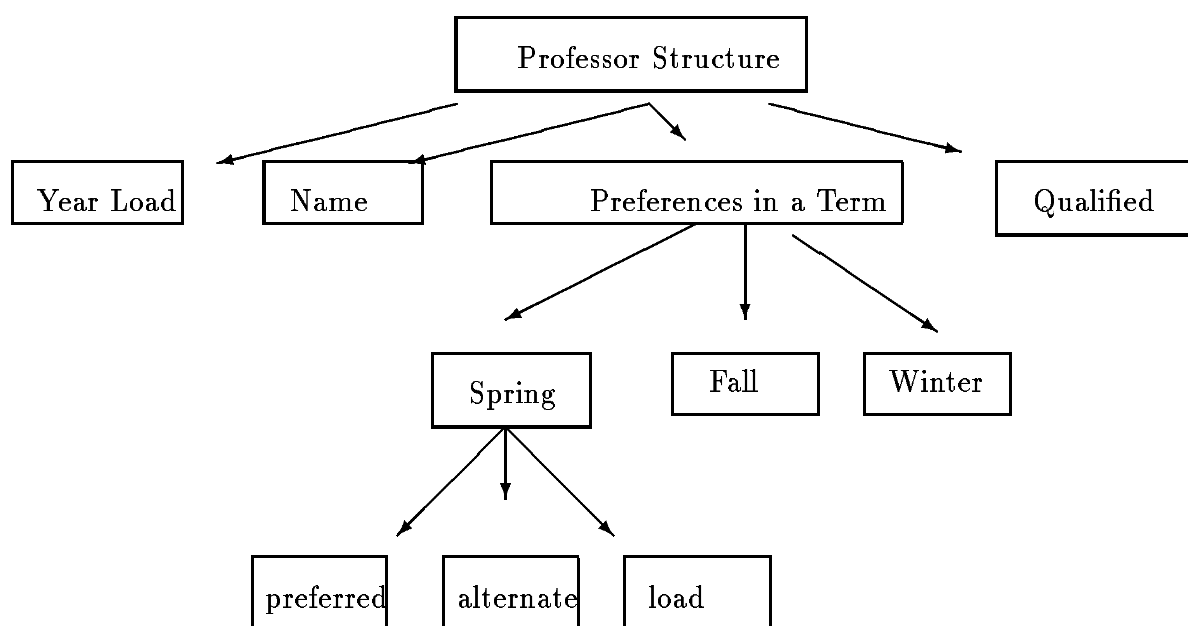

Figure 6: Professor data structure in the course scheduling domain.

Our course scheduling problem involves assigning professors to courses in one academic year, subject to a large number of constraints. Typically, the scheduling work is done by secretaries by hand, and is very time consuming. In the past, Operations Research methods have been implemented to address this problem, but success has been limited due to a number of reasons. First, using Integer Programming (IP) techniques to encode to domain requires converting constraints into weights, which is very hard to do properly even for experts. Second, secretaries would like to try adding and removing constraints during schedule generation, in order to verify how a schedule would compare with others. However, IP programs have to be changed each time this is done, and cannot meet the practical need of performing such *what-if* analysis in real time. Third, the course scheduling domain is typically revision oriented, which is not suitable for techniques that start from scratch. The change of a schedule from one year to the next may be restricted, but may still require modification due to changes in constraints, courses, professors, and student requirements. Thus, we have decided to solve the problem using our combined PCSP model. The result is a implemented scheduling system we call WATCOURSE.

The scheduling system, WATCOURSE, is implemented in Allegro Common Lisp on a Sun/4 workstation. It is domain independent in nature, and has been applied to the N-queens problem as well as the course scheduling problem. WATCOURSE has the ability to perform search either with or without abstraction, and has modules that can analyze a solution after it is found.

In the course scheduling problem domain, each professor is represented as a LISP struc-

ture, with a number of slots (See Figure 6). The course preferences of each professor is divided into three terms, Spring, Fall, and Winter, which correspond to the three teaching terms in a year. In each term, there is a list of preferred courses, alternative courses, and a preferred course load. In addition, each professor has a list of courses that he is qualified to teach. This information will be used to choose a course in case no preferred courses can be found for a professor.

The *demand-list* of a domain is a list of courses that are to be offered in one year. They include the number of sections a course will be offered in each term. For example, an item `((CS 486) Fall 2)` specifies that the course CS 486 will have two sections offered in Fall term. WATCOURSE uses this information to build a list of variables in the domain. each variable has the form:

```
(course-number term section-number)
```

For each variable, WATCOURSE builds a list of domain values based on the professors who are qualified to teach the course. In addition, a (null professor) value is associated with each course that can be cancelled due to tight constraints. That is, if a course X takes on the (null professor) value, then the course X is considered cancelled.

The constraints in our course scheduling domain is more complicated than those found in the N-queens domain. They include unary-constraints, binary-constraints, and k-ary constraints. Examples of the constraints are given below. Recall that a constraint takes as input a course-variable X, a professor v, and a remaining assignment A, and outputs a natural number representing the degree of violation of the constraint.

Unary Constraint: course preference. *If course X is preferred by professor v, then return 0, else return 1.*

Binary Constraint: exclusive courses constraint. *If a course X is offered in the same term as Y, then return 1, else return 0.*

3-ary constraint: year-offering constraint. *If a course X is not offered in A, and $v =$ (null professor), then return 1, else return 0.*

Meta Constraint. A meta constraint is one that is a constraint on the satisfaction of the rest of the constraints. For example, the following constraint is a meta one:

If for course X taught by professor v, neither constraint C_1 nor constraint C_2 is satisfied, then return 1, else return 0.

Note that our PCSP model for the course scheduling domain has the special advantage that certain hard constraints are implicitly satisfied by any solution. For example, consider the constraint that only one professor can teach a given section of a course in a given term. This constraint is always satisfied by solutions to the PCSP since, by definition, a variable can only take on a unique value in any solution.

The WATCOURSE program has been applied to course scheduling in our department, involving 119 courses and sections, and 40 professors. Comparing WATCOURSE with a

Linear Programming (LP) implementation for solving the same problem, we found that the LP program takes more than ten times longer than WATCOURSE to produce solutions of the same quality. We are currently still improving on the user-interface component of the system. But our testing so far has shown a reduction in scheduling time from days to less than one hour. What has been found to be particularly useful is the *any-time* feature of the system. That is, the system can be interrupted at *any time* to provide a solution. Although the solution may not satisfy all constraints, the more time is given to the system, the better is the solution quality. In addition, due to the random feature of the local search algorithm, every time the system is invoked, a different solution will be given. This feature is very useful for the system to work well as a consulting program, because when making the final decision, it is helpful to have several competing suggestions provided by the system.

10 Conclusions

In this paper, we have presented a model for partial solution of a constraint satisfaction problems. This model is a combination of the PCSP model and the local search technique. We have demonstrated that such a combination keeps all important features of the original methods. In addition, the combined system can naturally hand soft and hard constraints, solution revision, and hierarchical scheduling. We further demonstrated the feasibility of the approach by an application to a course scheduling domain.

In the future, we wish to further investigate hierarchical scheduling, as well as apply machine learning methods to solving partial constraint satisfaction problems with a local search method.

Acknowledgement

The authors wish to thank Jane Prime for her help in explaining the course scheduling domain, and to Nadia Benhessine for entering the course scheduling data.

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] Fahiem Bacchus and Qiang Yang. The downward refinement property. In *Proceedings of the 12th IJCAI*, pages 286–292, Sydney, Australia, August 1991.

[3] Mark Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. Morgan Kaufmann, 1987.

[4] Mark S. Fox, N. Sadeh, and C. Baykan. Constrained heuristic search. In *Proceedings of the 11th IJCAI*, pages 309–315, Detroit, Michigan, 1989.

[5] E. C. Freuder. Partial constraint satisfaction. In *Proceedings of the 8th AAAI*, pages 278–283, Boston, 1990.

[6] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction problems. Technical Report 92-01, Department of Computer Science, University of New Hampshire, Durham, NH, 03824, 1992.

[7] Craig Knoblock, Josh Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *Proceedings of the 9th AAAI*, Anaheim, CA, 1991.

[8] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1991. Tech. Report CMU-CS-91-120.

[9] Alan K. Mackworth. Consistency in networks of relations. In Webber and Nilsson, editors, *Readings in Artificial Intelligence*, pages 69–78. Morgan Kaufmann Publishers Inc., 1981.

[10] Alan K. Mackworth and Eugene C. Freuder. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 125:65–74, 1985.

[11] S. Minton, M. Johnston, A.B. Philips, and P. Laird. Solving large scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings of the 8th AAAI*, Boston, 1990.

[12] Earl Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[13] R. Sosic and J. Gu. A polynomial time algorithm for the n-queens problem. In *SIGART 1(3)*, 1990.

[14] D. Waltz. *Understanding Ling Drawings of Scenes with Shadows*. The Psychology of Computer Vision, ed. P.H.Winston. McGraw Hill, Cambridge, Mass., 1975.

[15] David Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm*. Morgan Kaufmann, CA, 1988.

[16] Qiang Yang and Josh Tenenberg. Abtweak: Abstracting a nonlinear, least commitment planner. In *Proceedings of the 8th AAAI*, pages 204–209, Boston, MA, August 1990.

[17] M. Zweben. A framework for iterative improvement search algorithms suited for constraint satisfaction problems. In *Technical report RIA-90-05-03-1, NASA Ames Research Center, AI Research Branch*, 1990.