Solving Incompletely Defined Constraint Satisfaction Problems with Nearest Neighbour Method

Yuiko Ohta and Nobuhiro Yugami
{yuiko,yugami}@flab.fujitsu.co.jp
Fujitsu Laboratories Ltd.

Abstract

This paper deals with incompletely defined constraint satisfaction problems (CSPs) with lacking part of constraints. Many real world problems can be formalized as CSPs, but their problem definitions are sometimes imperfect. Even if human experts know what kind of solutions are desirable, it is often difficult to describe their empirical knowledge in CSP forms explicitly. Moreover, complete problem definitions tend to be too complicated and make it costly to solve the problem.

This paper enhances traditional CSP techniques with k-nearest neighbour methods (k-NN). Our proposed CSP algorithm uses past solutions as implicit knowledge of the unknown parts of the problems. In many applications, human experts judge whether the solutions presented by CSP solvers are acceptable and modify them when they aren't desirable. It is natural to assume the accepted solutions are good enough on the unknown constraints. Our approach uses the cases as search control knowledge for both of variable and value selections, while it employs a tree search to solve CSPs. The experimental results show our algorithm works well for the incompletely defined problems.

1 Introduction

Constraint Satisfaction Problem (CSP) is a problem of finding an assignment of values for variables that satisfies given constraints. CSP has been applied to many real world applications, such as scheduling [Zweben, 1994], planning and product configuration [Sabin, 1998]. A variety of general algorithms [Tsang, 1993], [Mackworth, 1977], [Mackworth, 1985], [Minton, 1992], [Givry, 1997] have been developed for finding consistent, or less inconsistent assignments of values to predefined constraints.

However, in the real world applications, the problems are sometimes incompletely defined such that problem descriptions lack some constraints or priority of the constraints [Sycara, 1994], [Sqalli, 1998], [Biso, 2000]. Although human experts know what kind of solutions are desirable, it is laborious to describe their empirical knowledge precisely in CSP. Moreover, if the problem definition is perfectly defined, it tends to be too complicated and often makes to solve the problem costly. In such cases, usual CSP algorithms which only concern explicitly given constraints don't always find acceptable solutions because of the buried parts of problems.

We cannot avoid this difficulty if the problem to be solved is a completely new one and there is not any information to compensate for the lacking parts of the problem. However, many real world applications of CSPs require to solve similar problems repeatedly. For example, a production schedule of a factory is planned every week or every day with the same constraints on schedule and different orders of products to be manufactured. A manager usually checks the schedule presented by a scheduler (CSP solver) and modifies the schedule with his empirical knowledge when the schedule is not good enough with respect to the undefined parts of the problem. Then, it is natural to assume that the past accepted solutions implicitly represent the undefined parts of the problems and we can improve the solution to the current problem by using the past solutions.

K-nearest neighbour method (k-NN) [Dasarathy, 1991] is an interesting approach to utilize the instances of past problem solutions to solve the current problem. It has been mainly applied to classification problems. When a new instance to be classified is given, k-NN retrieves the k most similar instances from the past solutions and classifies the current problem to the most frequent solution (class label) among retrieved instances. Both of empirical [Michie, 1994] and theoretical results [Okamoto, 1997] showed the k-NN provides good performance with respect to prediction accuracy.

This paper proposes a new algorithm, CSP-NN, to solve CSPs with imperfect problem descriptions that uses k-NN to compensate unknown parts of the problems. CSP-NN doesn't use k-NN for directly solving CSPs, but uses k-NN as implicit search control knowledge for both of variable and value selections in tree search technique.

The remainder of this paper is organized as follows. In the next section, we review the previous works that use

past solutions to solve current CSPs. Next, we define the CSPs with imperfect problem descriptions and describe our proposed algorithm, CSP-NN. Finally, we show several empirical results to indicate the effectiveness of our algorithm.

2 Related Works

There are several researches and applications relating to compensating incomplete and incorrect problems by integrating CSP or optimization methods and Case-Based Reasoning.

[Sycara, 1994], [Miyashita, 1995] proposed a job shop scheduling systems, CABINS, which acquires context-dependent user optimization criteria by using Case-Based Reasoning (CBR). It stores cases as the set of values of features and the schedule repairing history features in advance. Repair history features represent which tactics is applied to improve the schedule, the effects of the repair, and whether the repaired schedule is acceptable or unacceptable. Once cases have been gathered, it generates an initial schedule by some heuristics and improves the schedule with its casebase. It invokes CBR to induce which action is to be applied to repair the current schedule and to estimate whether the new schedule is acceptable. They compared CABINS with simulated annealing (SA) to show the effectiveness and efficiency of CABINS. These experiments showed CABINS can acquire the different types of user's preference, and its effectiveness and efficiency were superior to SA on average.

[Sqalli, 1998] proposed a framework of enhancing CSP with CBR by giving an example of an application to diagnose the network problems. It uses CBR to infer what is missing or redundant in its model. It firstly employs CSP and after CSP fails to get a consistent answer, it invokes CBR to refine its problem model.

[Gervasio, 1999] represented an interactive system, INCA, to learn user evaluation functions. INCA assumes a weighted linear function for representing a user's preference in its model. INCA stores subordered evaluation of solutions from user's interactions and learns user's preference as a weighted linear function using a perceptron algorithm. They reported the predictive error of solutions generated by INCA to show its effectiveness. They assumed three users' evaluation functions, such as a weighted linear function, one with unknown features, and a nonlinear function, then they made INCA learn each type of users. Through empirical results, INCA can learn users' evaluation to supply good enough solutions for a weighted linear function and a nonlinear function. It was the most difficult function to learn a user evaluation function with unknown features, and the higher the ratio of unknown features to known ones was, the more difficult INCA adapted to the user's preference. For all evaluation, INCA improved the performance as the number of training examples increased.

[Biso, 2000] proposed an algorithm to learn soft constraints and extended the algorithm to an incremental one. It learns the soft constraints by the gradient decent technique with the set of instances which consists of solutions and their evaluation. They reported its effectiveness and efficiency through experimental results with the noisy instances.

These researches showed the effectiveness of utilizing past instances for compensating the missing parts of problem definitions and the former three focused on certain applications. The first one uses the casebase to estimate the users' preference dynamically on the job-shop scheduling problems and it improves the solution iteratively. The latter two use the casebase to refine the problem models explicitly.

The work in this paper is a similar approach to these researches in respect of using past instances, but it doesn't intend to generate an explicit problem model, but uses the instances as (implicit) search control knowledge directly. Then, CABINS is the more similar to our approach. However, our approach uses a final solution for each past problem as a case, whereas CABINS uses each improvement step in problem solving as a case. In addition, this paper deals with general CSPs instead of some specific applications.

3 Problem Description

This section illustrates CSPs with imperfect problem descriptions (Imperfect CSPs) dealt with in this paper.

An imperfect CSP is given by a set of variables, a set of possible values (domain) for each variable, a partial value assignment for some of the variables that represents a current problem to be solved, a set of constraints that restrict combinations of values of plural variables, and a set of instances (casebase) that represents the results of past problem solving.

As described in introduction, we assume a situation that requires to solve similar CSPs with the same constraints repeatedly. To represents such a situation, we divide variables into two kinds, conditional variables and solution ones. The conditional variables represent the current problem to be solved and their values are fixed and given to CSP solvers. The solution variables represent a solution of the problem defined by the value assignment of conditional variables and the values for the solution variables are not given. Solving an imperfect CSP is to find an assignment of values for solution variables depending on the values of the conditional variables.

Imperfect CSP assumes missing parts of problem description such as constraints and preference of solutions. Therefore it is not sufficient for getting an acceptable solution to satisfy all of the given constraints. Instead, imperfect CSP is given instances of past problem solving as assignments of values for both of conditional and solution variables that represent pairs of past problems and their sufficiently good and acceptable solutions. The purpose of CSP solver is to find an assignment for the solution variables that not only satisfy all of the given constraints but also is good with respect to the missing constraints and preference of solutions embedded in the

Table 1: Pseudo-code of CSP-NN : At first call, **Sol** is an assignment of values for conditional variables.

CSP-NN (**Sol**)
/* **Sol** : a set of values already assigned to */
1: `If` **Sol** violates any constraint,
return **nil**.
2: `If` there is no unlabelled solution variable,
return **Sol**.
3: Retrieve the k most similar instances to **Sol** from the casebase.
4: Select variable x which maximizes Eq. (1) from unlabelled solution variables.
5: Sort values of the selected variable x in descending order of the frequency among the retrieved instances.
6: `For each` value v in the value order list from step 5,
7: **Sol'** $\leftarrow$ **CSP-NN** (**Sol** $\cup$ $(x = v)$).
8: `If` **Sol'** $\neq$ **nil**,
return **Sol'**.
9: Return **nil**.

instances.

In the production scheduling, the conditional variables represent the required quantities and due dates for products and they are given before scheduling. The solution variables are for a schedule to produce the required products defined by the conditional variables. The daily (or weekly) schedules are generated by solving a CSP, with different value assignments to the conditional variables, but with the same set of variables, domains and constraints.

In the following of this paper, we only consider discrete CSPs in which each domain is a finite set of discrete values.

4 CSP solver with k-NN

In this section, we describe our CSP algorithm enhanced by k-NN for imperfect CSPs. As mentioned above, we assume our CSP solver has a casebase. An instance in the casebase is assignments of values for both of conditional and solution variables, representing the problem and its solution.

Table 1 shows the outline of our algorithm, CSP-NN.

CSP-NN employs depth-first tree search algorithm. At the start, an assignment of values for conditional variables is given to CSP-NN. Then, it assigns a value to one unlabelled variable at a time until values of all variables are assigned. When the search fails before labelling all variables, CSP-NN backtracks and tries a different assignment. The performance of tree search technique basically depends on how to select a variable whose value is assigned next and how to select a value for the selected variable. CSP-NN uses nearest neighbour method in these selections. At each node of a search tree, CSP-NN retrieves k instances from the casebase which are similar to the current partial assignment and counts frequencies of values in the retrieved instances.

Because the retrieved instances show the good assignments with similar problems which are represented as the assignment for the conditional variables and similar partial solutions, then it can be expected that a value with high frequency among the retrieved instances tends to constitute a better solution than a value with low frequency. If only one value for a certain variable has high frequency and the frequencies of the remaining values are quite low, selecting the value for the variable leads a good solution with high probability. In the opposite, if the frequency is uniform for all values for a certain variable, there is no information to decide which value is to be selected for the variable and it is better to assign its value after values of other variables are selected.

Based on this discussion, at step 4 in Table 1 CSP-NN selects a variable that maximizes the following evaluation function.

$$\sum_{v \in D'(x)} \left(\frac{p(x,v)}{\sum_{v \in D'(x)} p(x,v)} \right)^2 \qquad (1)$$

where $p(x,v)$ is the frequency of instances which value of the variable x is v among k retrieved instances, and $D'(x)$ is the set of values for variable x which are consistent with the current partial assignments.

After selecting the variable to be labelled next, CSP-NN assigns the value v for the selected variable with descending order of the frequency, $p(x,v)$.

CSP-NN retrieves similar instances to the current partial assignments **Sol** by k-NN at each node of a search tree (step 3 in Table 1) and the complexity of this process is about the product of the number of variables and the number of the instances in the casebase. To decrease this complexity, CSP-NN stores the distances between the current **Sol** and all instances and reuses them after the value of the next variable is selected at step 7 by updating the distances with only the contributions of the variable. In many applications of k-NN, the distance measures is a weighted sum of a distance of values of every variable. In such a case, the complexity of k-NN becomes only the number of instances and is independent of the number of variables excluding a root node.

For brief explanation, Table 1 assumes simple depth first search. However, CSP-NN can employ for more sophisticated tree search technique such as best first search and intelligent backtracking with small modification. Furthermore, when checking consistency of values with the current assignment and detecting constraints at step 1, CSP-NN can work with constraint propagation technique such as forward checking and arc consistency algorithms.

5 Experiments

This section empirically verifies the performance of CSP-NN, how the resulting solution reflects the missing criteria, how many instances are required to obtain reasonable solutions, how the quality of the casebase affects

the goodness of the solution, and how nearest neighbour retrieval increases the time complexity.

In all experiments, we use binary soft constraints as the missing parts of the problems. A soft constraint assigns a score from 1 to 10 randomly for every value pair of two variables in the constraint. The soft constraints are almost similar to the criteria $\sum -VCSP$ in [Givry, 1997]. The goodness of the solution is measured as the total amount of the scores of values pairs of the soft constraints. All of the soft constraints are unknown to the solver.

Through the experiments, we generated CSPs randomly with six parameters, the number of conditional variables, P, the number of solution variables, N, the domain size of each variable, D, the number of binary hard constraints, C, the tightness of the constraint, t, and the number of binary soft constraint, U. In the experiment, we fixed $D = 4$, and $t = 0.3$. This means that every variable has four possible values and every hard constraint prohibits each of sixteen value pairs with probability 30 %. We verified CSP-NN by binary constrained problems in this paper, but CSP-NN works in the same manner for not-binary constraints.

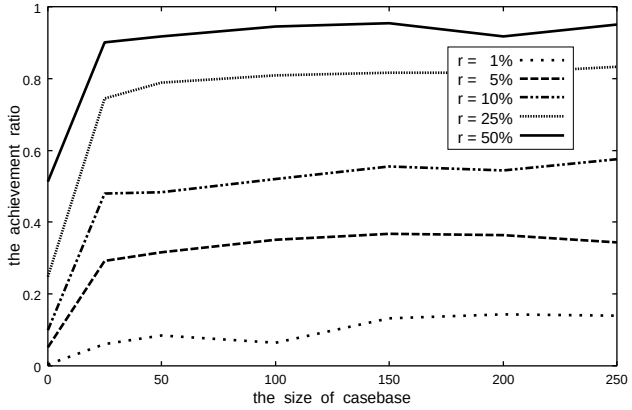
We used hamming distance as a distance measure to retrieve instances. When several instances have the same distance from the current assignment of values, the order of similarities among them to the current assignment is determined randomly. We fixed $k = 10$ in the experiments.

We tried two kinds of casebase. Both type of casebases are generated by solving randomly generated problems. One casebase (Type 1) consists of the instances searched by hill climbing method, and the instances in the casebase were not always the optimum but were expected good solutions. The other (Type 2) was generated by strict optimization and it consists of optimum solutions.

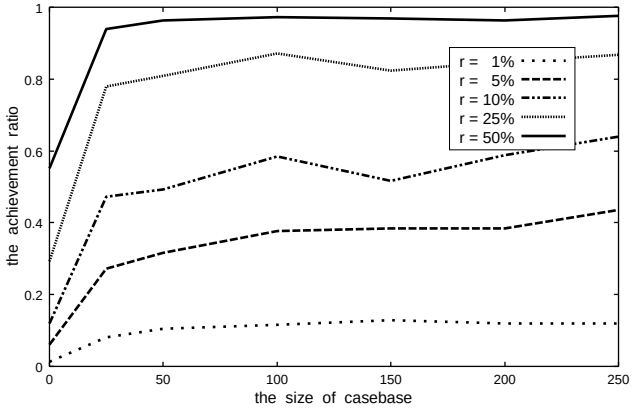
To evaluate the quality of a solution, we use the rank of the solution to be evaluated among the all consistent solutions that satisfy all of the hard constraints. For example, if the solutions obtained by CSP-NN are involved in top 10 % solutions with higher probability than 10 %, then we can judge CSP-NN can effectively utilize past solutions to solve the current problem.

To verify the performance clearly, we give the solver the assignment for the conditional variables such that they have consistent solutions with hard constraints for each trial. During testing, all results were based on 200 trials generated with 10 problems and 20 different initial assignments to the conditional variables for each problem.

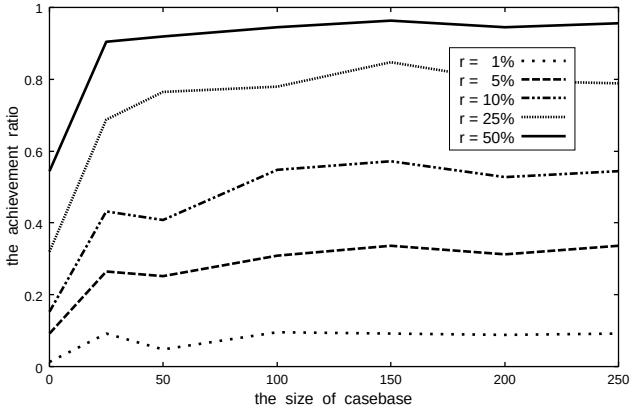
At first, we show how the goodness of solutions improves depending on the size of casebase. In these experiments, CSP-NN uses the Type 1 casebase which instances were generated by hill climbing search. Figure 1 shows the ratio to total trials which the solutions achieve with each solution rank, r. Each figure in Figure 1 is for the problems with sparse soft constraints $U = 30$, medium $U = 60$ and dense $U = 100$, respectively.



(a) sparse soft-constraints $U = 30$



(b) medium soft-constraints $U = 60$



(c) dense soft-constraints $U = 100$

Figure 1: Dependency of the goodness of solutions of CSP-NN for the size of Type 1 casebase : $N = 16$, $P = 4$, and $C = 30$

When the size of casebase is 0, it returns a solution randomly among solutions which are consistent to the hard constraints and CSP-NN with casebase size = 0 corresponds to traditional CSP solver with depth first search.

In Figure 1, the achievement ratios of each solution rank make significant improvement as increase of casebase size, independent of the density of soft-constraints. When casebase involves more than 50 instances, CSP-NN could output solution better than the average of the consistent solutions at 90 % of trials, and could output best 10 % solutions at over a half of trials. In terms of the casebase size, CSP-NN with a very small casebase cannot find good solutions with high probability. However, there is marginal ability in growing the goodness of solutions according as the increase of the size of casebase. In the cases of Figure 1, when the size of casebase is between 100 and 150, the goodness of solution seems to be saturated.

Figure 2 shows the result for the dense problem with more hard constraints than the problem in Figure 1(b). For dense hard constrained problems, the effect of CSP-NN is smaller than for sparse problems. This is because the dense constraints decrease the number of satisfiable solutions, and they enlarge the distances between the solution and similar instances retrieved from the casebase. Then, the dense problem causes the worse solution.

Nevertheless, CSP-NN works effectively for the imperfect CSPs with dense hard constraints when the number of instances is 100 or more, and the necessary size of casebase for it is relatively small to obtain reasonable solution.

Next, we show how the solution is influenced by the quality of the casebase.

Figure 3 shows the results from two types of casebase. Figure 3(a) is the result from the Type 1 casebase which instances are generated by hill climbing method and Figure 3(b) is from the Type 2 casebase which consists of optimal solutions.

With Type 2 casebase, the goodness of solutions also develops with the relatively small size of casebase, as with Type 1 casebase. Moreover, the achievement ratio with Type 2 casebase makes much more improvement than with Type 1 casebase, especially for the small solution rank. The solver with Type 2 casebase can find best 1% solutions at a half of trials with 50 instances, while the solver with Type 1 casebase hardly finds ones. Likewise, the solver with Type 1 casebase finds best 25 % solutions with the 90 % probability, the one with Type 2 casebase can find best 10 % with the same probability.

This experiment indicates if it is possible to gather the quite good past solutions, the solutions obtained by CSP-NN are expected to be excellent. However, CSP-NN can find good enough solutions with not quite good casebase.

Lastly, we examine the time complexity of our algorithm. CSP-NN employs k-NN at every variable-

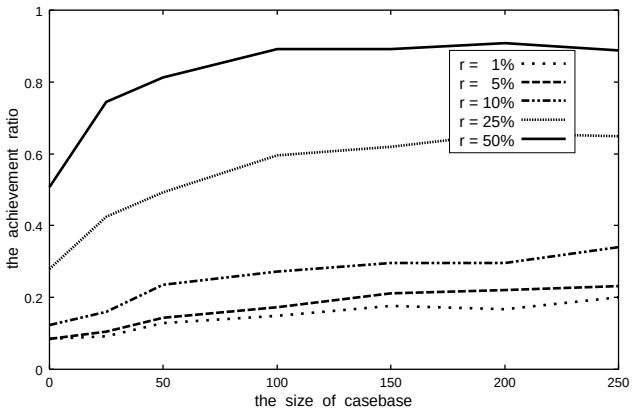
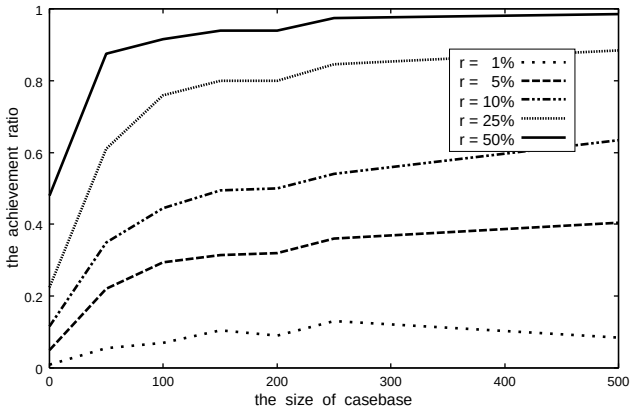
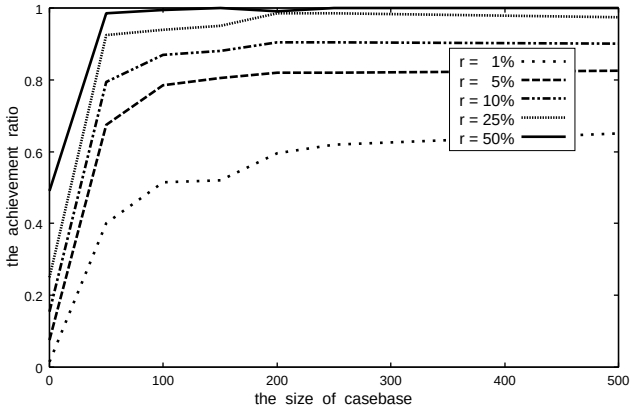


Figure 2: The goodness of solutions of CSP-NN. For the dense constrained problem : $N = 16$, $P = 4$, $C = 100$, and $U = 60$



(a) with Type 1 casebase



(b) with Type 2 casebase

Figure 3: Dependency of the goodness of solutions of CSP-NN for the quality of the casebase : $N = 20$, $P = 5$, $C = 100$, and $U = 100$

Table 2: Average of the CPU time and the solution rank with Type 1 casebase and executed on a Sun workstation with 300MHz UltraSPARC

(a) $N = 16,\ P = 4,\ C = 30$, and $U = 60$ sparse hard-constraints

size of casebase	CPU time (msec.)	solution rank
0	17.6	48.2%
50	21.9	14.8%
100	22.9	12.0%
250	24.0	11.6%

(b) $N = 16,\ P = 4,\ C = 100$, and $U = 60$ dense hard-constraints

size of casebase	CPU time (msec.)	solution rank
0	24.4	49.9%
50	27.2	33.8%
100	27.8	28.6%
250	28.3	24.5%

(c) $N = 20,\ P = 5,\ C = 100$, and $U = 100$ medium hard-constraints

size of casebase	CPU time (msec.)	solution rank
0	24.7	50.2%
50	30.3	23.9%
100	31.2	18.0%
500	31.4	11.0%

selecting step, so it may cause a significant drop in efficiency of CSP-NN. Table 2 reports the average of the CPU time and the solution ranks.

These results show CSP-NN doesn't make the calculation time so worse in comparison with the one not-using the casebase (casebase size = 0) and the increase of the casebase size doesn't degrade so much in time complexity. We used hamming distance as distance measure in this section and the time complexity of calculating distances for all instances is $O((P+N)\times$ casebase size) at each node in search tree. However, as mentioned in the previous section, CSP-NN reuses the results at each node in its child nodes and can decrease the complexity to $O($ casebase size) when it has not backtracked. As a result, CSP-NN is not so slow compared with traditional tree search algorithm. Empirical results confirm it is effective to save the calculation time and CSP-NN with the distance measure which assumes the independence among variables provide good solution in reasonable time.

The above all experiments show CSP-NN can provide a good solution in a reasonable time for the imperfect CSPs.

6 Conclusion

We have presented a CSP algorithm, CSP-NN, that uses past solutions to solve the incompletely defined problems. CSP-NN employs k-NN and uses the retrieved instances as search control knowledge for variable and value selections.

Experimental results showed this algorithm could obtain good solutions for the problem missing soft constraints in comparison with the CSP having no casebase. The results also indicated the appropriate size of casebase was not so large and retrieving similar instances degrades time complexity slightly. CSP-NN can find a good enough solution in reasonable search time.

Even if we only verify the algorithm with CSPs with missing soft constraints, we expect this algorithm has a potential to solve other types of incompletely described CSPs with missing hard constraints and incorrectly defined constraints. CSP-NN may treat such problems only with extending the evaluation functions to select a variable and a value.

We verified CSP-NN only with randomly generated problems with uniform distribution. However, the distribution is not uniform in most of real world problems. We need to explore the effectiveness of the algorithm with problems generated with more realistic distribution.

References

[Mackworth, 1977] Alan K. Mackworth. Consistency in Networks of Relations. *Artificial Intelligence* , 81(1):99–118, 1977.

[Mackworth, 1985] Alan K. Mackworth, and Eugene C. Freuder. The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems. *Artificial Intelligence* , 25(1):65–74, 1985.

[Dasarathy, 1991] Belur V. Dasarathy. *Nearest Neighbor(NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, 1991.

[Minton, 1992] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Phillip Laird. Minimizing Conflicts: A Heuristic Repair Method for Constraint-Satisfaction and Scheduling Problems. *Artificial Intelligence* , 58:161-205, 1992.

[Tsang, 1993] , Edward Tsang. *Foundations of Constraint Satisfaction.* Harcourt Brace & Company, 1993.

[Zweben, 1994] , Monte Zweben, and Mark S. Fox. *Intelligent Scheduling.* Morgan Kaufmann, 1994.

[Sycara, 1994] Kaita Sycara and Kazuo Miyashita. Case-Based Acquisition of User Preferences for Solution Improvement in Ill-Structured Domains. In *Proceeedings of the Twelfth National Conference on Artificial Intelligence* , 1994.

[Michie, 1994] Donald Michie, David J. Spiegelhalter, and Charles C. Taylor. *Machine Learning, Neural and Statistical Classification.* Fllis Horwood, 1994.

[Miyashita, 1995] Kazuo Miyashita and Katia Sycara. CABINS:A Framework of Knowldge Acquisition and

Interative Revision for Schedule Improvement and Reactive Repair. *Artificial Intelligence*, 76(1–2):377–426, 1995.

[Givry, 1997] Simon de Givry, Gérard Verfaillie, and Thomas Schiex. Bounding the Optimum of Constraint Optimization Problems. In *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming*, 405–419, 1997.

[Okamoto, 1997] Seishi Okamoto and Nobuhiro Yugami. An average case analysis of the k-nearest neighbor classifier for noisy domains. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 238–243, 1997.

[Sqalli, 1998] Mohammed H. Sqalli and Eugene C. Freuder. Diagnosing InterOperability Problems by Enhancing Constraint Satisfaction with Case-Based Reasoning. In *Proceedings of the Ninth International Workshop on Princples of Diagnosis*, 266–273, 1998.

[Sabin, 1998] Daniel Sabin and Rainer Weigel. Product Configuration Frameworks – A Survey. *IEEE Intelligent Systems*, 13(4):42–49, 1998.

[Gervasio, 1999] Melinda T. Gervasio, Wayne Iba, and Pat Langley. Learning User Evaluation Functions for Adaptive Scheduling Assistance. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 152–161, 1999.

[Biso, 2000] Alessandro Biso, Francesca Rossi, and Alessandro Sperduti. Experimental Results on Learning Soft Constraints. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, 2000.