Report Number: WUCS-01-01

2001-01-01

Local Search and Encoding Schemes for Soft Constraint Minimization Problems

Michael P. Moran and Weixiong Zhang

Soft constraint minimization problems (SCMPs) contain hard constraints that cannot be violated and soft constraints that may be violated but carry penalties if not satisfied. In this paper, we first extend local search, WalkSAT in particular, to SCMPs and study the existing SAT encoding schemes for SCMPs. We propose a general encoding method called k-encoding. We then investigate the effects of local search neiborhood structures introduced by encoding schemes and analyze the anytime performance of extended WalkSAT using different encoding methods. Our experimental results on various graph coloring problems show that a direct extension of WalkSAT is most effective,... **Read complete abstract on page 2.**

Local Search and Encoding Schemes for Soft Constraint Minimization Problems

Michael P. Moran and Weixiong Zhang

Complete Abstract:

Soft constraint minimization problems (SCMPs) contain hard constraints that cannot be violated and soft constraints that may be violated but carry penalties if not satisfied. In this paper, we first extend local search, WalkSAT in particular, to SCMPs and study the existing SAT encoding schemes for SCMPs. We propose a general encoding method called k-encoding. We then investigate the effects of local search neiborhood structures introduced by encoding schemes and analyze the anytime performance of extended WalkSAT using different encoding methods. Our experimental results on various graph coloring problems show that a direct extension of WalkSAT is most effective, and that WalkSAT using binary encoding is particularly ineffective. Our study also shows that encoding may provide special neighborhood structures that can speed up local search algorithms.

Local Search and Encoding Schemes for Soft Constraint Minimization Problems

Michael P. Moran and Weixiong Zhang

WUCS-01-01

November 2000

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130

Local Search and Encoding Schemes for Soft Constraint Minimization Problems*

Michael P. Moran and *Weixiong Zhang*†
Department of Computer Science
Washington University
St. Louis, MO 63130

January 21, 2000

Abstract

Soft constraint minimization problems (SCMPs) contain hard constraints that cannot be violated and soft constraints that may be violated but carry penalties if not satisfied. In this paper, we first extend local search, WalkSAT in particular, to SCMPs and study the existing SAT encoding schemes for SCMPs. We propose a general encoding method called k-encoding. We then investigate the effects of local search neighborhood structures introduced by encoding schemes and analyze the anytime performance of extended WalkSAT using different encoding methods. Our experimental results on various graph coloring problems show that a direct extension of WalkSAT is most effective, and that WalkSAT using binary encoding is particularly ineffective. Our study also shows that encoding may provide special neighborhood structures that can speed up local search algorithms.

*This research was partially supported by NSF grants IRI-9619554 and IIS-0196057 and DARPA Cooperative Agreement F30602-00-2-0531.

†Corresponding author: Weixiong Zhang, Department of Computer Science, Washington University, Campus Box 1045, One Brookings Drive, St. Louis, MO 63130. email: zhang@cs.wustl.edu

1 Introduction and Overview

One of the most important recent advances in the studies of constraint satisfaction problems (CSPs) is the introduction of soft constraints [1, 5, 12, 14] to capture many important aspects of real-world constraint problems.

In traditional CSPs, all constraints are treated equally, and must all be satisfied to solve the problem. Such CSPs are not always effective at modeling real-world constraint problems. *Soft constraint* is a categorical term that may represent such constraints that need not necessarily be satisfied but carry penalties if violated or provide rewards if satisfied. Examples of soft constraints include constraints with associated importance, costs, user preference, and uncertainty. A real-world constraint problem is typically a combination of soft constraints and constraints that absolutely cannot be violated, called *hard constraints*. We refer to a CSP with hard and soft constraints as soft CSP (SCSP) [1, 2], which, we believe, is a more realistic model for practical constraint problems.

More importantly, SCSP makes the study of over constrained problems much more meaningful. There are many real-world applications for which not all constraints can be satisfied at the same time. Take as an example a distributed resource allocation problem in multi-agent systems, a problem that motivated our research on SCSP. In this problem, multiple agents must control, allocate, schedule and coordinate various sensors and actuators to perform a complex task. A single step of the task may use different resources, i.e., sensors and actuators. In cases when multiple resources can be chosen, different sensors and actuators may give different real-time performance and use different amount of energy. Therefore, the problem can be formulated as a SCSP with soft constraints carrying a weight representing the operational condition of a resource. One of the important features of the problem is that it is typically *over constrained*, meaning that there may not exist a variable assignment to satisfy all constraints.

In an over constrained situation, the goal is to find a variable assignment to satisfy all the hard constraints as well as minimize the overall weight of the violated soft constraints. In other words, SCSP is an optimization problem. In contrast, CSP is a decision problem in the sense that when it is over constrained, reporting that no solution is possible is sufficient.

Many issues regarding the representational power and applications of SCSP have been considered. See [2] for an excellent survey and tutorial. However, the study of the computational issues of SCSP is limited. The only published related work that we are aware of is a study of depth-first branch-and-bound (DFBnB) on over constrained CSP, a special case of SCSP [6, 17]. DFBnB is a systematic search algorithm which constructively enumerates possible solutions, and consequently is unable to solve large problems within a reasonable amount of time.

In this research, we are interested in the computational issues of SCSP. Inspired by the great success of local search for Boolean satisfaction (SAT) problems [8, 16],

we apply local search to SCSP. Our goal is to extend effective local search algorithms for SAT, such as the WalkSAT algorithm [15], to SCSP. Unlike a systematic search, a local search is often able to solve large SAT problems of several thousand variables.

There are at least two critical issues that need to be addressed in order to extend a local search algorithm for SAT to an efficient local search algorithm for SCSP. The first issue concerns soft constraints. In a SCSP, soft constraints may represent different types of constraints, such as importance, costs and uncertainty. How these different types of constraints compare to one another relates to how a problem should be modeled as a SCSP, an issue beyond the scope of this paper.

To facilitate our investigation and reflect the computational difficulty of SCSP, we consider *soft constraint optimization problem* that have constraints with weights categorically representing the weights of soft constraints for which being importance, costs, or uncertainty. In other words, we assume in this paper that a mechanism is given to compare two constraints of different types. Formally, a soft constraint optimization problem can be specified by a set of variables and a set of constraints over the variables. Each variable has its domain, a set of possible values that the variable can take. Each constraint is a Cartesian product of the domains of a set of variables, which specifies the allowed or forbidden combinations of variable values. Furthermore, a constraint can be a hard constraint that cannot be violated, or a soft constraint that carries a penalty if it is violated (or alternatively carries a reward if the constraint is satisfied). Such a penalty (reward) is represented as a weight of the constraint. The solution to a soft constraint optimization problem is an assignment of values to their variables so that the overall weight of the unsatisfied (satisfied) constraints is minimized (maximized). In our study, we consider *soft constraint minimization problems* (SCMP).

The second issue in extending local search to SCMP concerns variable domains. In a local search algorithm for SAT, a single variable can be "flipped" efficiently to change its value from True to False or vise versa. Unlike the Boolean variables in SAT, the domains of SCMP variables may be large, and hence there are many value options when a variable must be changed. A local search algorithm for SCMP must therefore decide not only which variable to change but which value to change to.

Recent studies on different encoding methods have suggested that mapping a CSP to a SAT may be useful to local search algorithms [4, 7, 9, 10]. On the other hand, all these studies are restricted to CSP, which is a problem of deciding if a problem is satisfiable or not. The SCMP framework that we consider in this research is an optimization problem, i.e., when the problem is unsatisfiable, a solution that minimizes the total weight of violated constraints is required. This optimization problem is computationally more difficult than its decision counterpart. It is not clear if encoding a problem to SAT is more effective than solving it with a direct extension of the "flipping" operation.

To answer this question, we study different encoding schemes in this paper. We first examine the existing SAT encoding schemes, which encode each SCMP variable

with a set of Boolean variables [4, 7, 9, 10]. To further understand the effects of encoding, we propose a new encoding method called k-encoding. Briefly, k-encoding represents a variable of a large domain with a set of variables each of which takes at most k values. K-encoding takes binary encoding as a special case.

Our experimental study shows that the main effect of k-encoding on local search is to restrict the possibilities of value selection. It imposes a special structure onto the values of a variable, which limits the number of neighboring states that can be examined in each step of a local search. Based on this observation, we directly restrict the neighboring states that a local search algorithm can evaluate, and compare it against k-encoding. The results from this comparison significantly deepens our understanding of encoding methods for local search.

We use graph coloring as our problem domain in our experiments. We tested our extended WalkSAT algorithm on randomly generated problem instances and instances of real CSP formulated as graph coloring [3].

The paper is organized as follows. In Section 2, we discuss different encoding schemes, including unary encoding, binary encoding, and our new k-encoding. In Section 3, we describe the WalkSAT algorithm and consider how it can be extended to SCSP. We also examine the local search neighborhoods of different encoding schemes. We present the experimental results on graph coloring problems in Section 4. We discuss related work in Section 5. Finally conclude in Section 6 and discuss future work in Section 7.

2 Encoding Schemes

As mentioned in Section 1, local search algorithms are very efficient at solving SAT problems. The objective of an encoding is to map variables with large domains to variables with smaller domains. Presumably local search algorithms can then be applied to find solutions quickly.

In this section, we describe two existing SAT encoding methods and propose our new encoding scheme. In our discussion, we use a running example to illustrate each of the encoding methods. Our simple example has two variables, X with domain $D = \{d_0, d_1, \cdots, d_{m-1}\}$, and Y with domain $D' = \{d'_0, d'_1, \cdots, d'_{m'-1}\}$. The constraints between X and Y are represented as nogoods, which are forbidden value pairs between the variables. Denote the set of nogoods as N, then $N = \{(d_i, d'_j) | \text{where } d_i \in D, d'_j \in D',$ and (d_i, d'_j) is a forbidden value pair$\}$.

2.1 SAT encodings: Existing schemes

A SAT encoding maps a SCMP with large variable domains to a SAT problem with Boolean variables, i.e., variables with values T or F. Two SAT encoding schemes have been proposed and studied. One is the unary encoding and the other the binary

encoding. In exchange for reducing their domain sizes, both schemes increase the number of variables, and often introduce additional hard constrains to the SCMP representation.

2.1.1 Unary encoding

The unary encoding creates one new Boolean variable for each value in every variable's domain. Therefore every variable with domain size m corresponds to a set of m Boolean variables. Additional hard constraints are introduced in order to force exactly one Boolean encoding variable to be true.

In our running example, we associate a Boolean variable, x_i, with each value d_i of variable X, and $x_i = T$ if X takes value d_i and $x_i = F$ otherwise, for $i = 0, 1, \cdots, m-1$. We then use a clause to ensure that X takes a value, i.e., $x_0 \vee x_1 \vee \cdots \vee x_{m-1}$. We also have clauses that ensure X takes no more than one value, i.e., for each i, j with $i \neq j$, $\neg x_i \vee \neg x_j$. We can similarly encode variable Y with a set of Boolean variables, $y_0, y_1, \cdots, y_{m'}$. Finally, we introduce clauses equivalent to the constraints between X and Y. For nogood (d_i, d'_j), we have the clause $\neg(x_i \wedge y_j) = \neg x_i \vee \neg y_j$.

While unary encoding has shown some success in solving CSP problems, an encoded problem can be significantly larger than the original problem. In particular, when the sizes of variable domains are large, the number of encoding variables and number of constraints introduced by the encoding can be prohibitively large.

2.1.2 Binary encoding

In binary encoding, every variable in a SCMP is represented by a set of Boolean variables in the encoded problem. A variable with domain size m is represented by $\lfloor \log m \rfloor$ new variables in the encoded problem. This set of Boolean encoding variables, in combination, can represent any value of the original variable. Because each variable is replaced with $\lfloor \log m \rfloor$ new encoding variables, the encoded problem will have more variables and more constraints than the original problem. If the size of a variable's domain is not a power of two, then the $\lfloor \log m \rfloor$ new variables will over represent the unencoded variable and allow the encoded problem to take a state that has no meaning in the original problem. In such a case, additional hard constraints, which cannot be violated, need to be added in order to forbid certain combinations of the new Boolean variables.

In our example, the domain of X, $D = \{d_0, d_1, \cdots, d_{m-1}\}$, can be represented by $\lceil \log_2 m \rceil$ Boolean variables, x_0, x_1, x_{l-1}, where $l = \lceil \log_2 m \rceil$. For instance, variable X with domain size four can be encoded by two Boolean variables x_0 and x_1, and its domain $D = \{d_0, d_1, d_2, d_3\}$ can be represented by $\{d_0 = \neg x_1 \wedge \neg x_0, d_1 = \neg x_1 \wedge x_0, d_2 = x_1 \wedge \neg x_0, d_3 = x_1 \wedge x_0\}$. If $\lceil \log_2 m \rceil > \log_2 m$, we also introduce clauses to rule out spurious values due to the encoding. For example, if X has only three values $\{d_0, d_1, d_2\}$, we have clause $\neg(x_1 \wedge x_0) = \neg x_1 \vee \neg x_0$ to exclude the spurious

value d_3. Variable Y with domain $D' = \{d'_0, d'_1, \cdots, d'_{m'-1}\}$ can be similarly encoded by $\lceil \log_2 m' \rceil$ Boolean variables, $y_0, y_1, \cdots, y_{l'-1}$, where $l' = \lceil \log_2 m' \rceil$. We further introduce a clause for each nogood between X and Y. For nogood (d_1, d'_0) for instance, we have clause $\neg(\neg x_1 \wedge x_0 \wedge \neg y_1 \wedge \neg y_0) = x_1 \vee \neg x_0 \vee y_1 \vee y_0$.

2.2 K-encoding: A general scheme

Notice that binary encoding discussed in the previous section is simply an encoding with base 2. Similarly, we develop a general encoding with base k, which we call k-encoding.

In *k-encoding*, a variable is encoded by a set of variables with domain no larger than k, where k is less than the domain size of the original variable. For example, if $k = 3$, a variable of domain size nine can be represented by two variables, x_1 and x_0, each with a domain of size three, $\{0, 1, 2\}$ for instance. Specifically, domain $D = \{d_0, d_1, d_2, \cdots, d_8\}$ can be represented by $\{d_0 = ((x_1 = 0) \wedge (x_0 = 0)), d_1 = ((x_1 = 0) \wedge (x_0 = 1)), d_2 = ((x_1 = 0) \wedge (x_0 = 2)), \cdots, d_8 = ((x_1 = 2) \wedge (x_0 = 2))\}$.

Using k-encoding, a variable of domain size m is represented by $\lceil \log_k m \rceil$ variables of domain size k each. Similar to binary encoding, we then introduce clauses to encode the nogoods of the original problem.

Note that binary encoding is a special case of k-encoding with $k = 2$. For the rest of this paper, we refer to specific k-encodings by replacing k with an integer value. Binary encoding is therefore called 2-encoding.

3 Local Search for Soft Constraint Minimization

This section discusses how a local search algorithm for SAT can be extended to SCMP. We specifically describe extending the WalkSAT algorithm. We also consider the local search neighborhoods of WalkSAT under different encoding schemes.

3.1 WalkSAT for SCMP

The local search algorithm used in our tests is an extension of the WalkSAT algorithm for SAT problems. The conventional WalkSAT-G algorithm [11] progresses by randomly choosing an unsatisfied constraint, and then "flipping" the Boolean variable in that constraint that will result in the greatest number of satisfied clauses. WalkSAT-G flips a randomly selected variable in the clause with probability p. Our generalized WalkSAT also chooses an unsatisfied constraint at random, but then considers changing every variable in the clause to each value in its domain and eventually changing the variable with a value that does not violate hard constraints and minimizes the weight of the resulting unsatisfied soft constraints. Changing a variable value that reduces the total number of unsatisfied constraint is call a downward move,

or an upward move otherwise. Since one variable must be changed at every move, the algorithm may make upward moves when no downward moves are possible.

In order to determine which variable and value should be changed in order to minimize the weight of the resulting unsatisfied clauses, the effect on all of the clauses must be considered for every variable-value pair in the clause. When a value change is being evaluated, it is first checked against any hard constraints. If any hard constraint is violated in the neighboring assignment, the move is immediately rejected. If all of the hard constraints are satisfied, then the effects of the change are evaluated on each of the soft constraints in which the changed variable is involved.

The cost of evaluating all neighboring assignments for the unsatisfied clause is a function of the number of variables in the clause or the satisfiability of the clauses plus the total number of neighboring assignments evaluated, since the effects from changing a variable involved are determined for both its current value and all of it's neighboring values. Other implementations of WalkSAT may take time that is proportional only to number of neighboring assignments evaluated, but we believe our current implementation is more efficient.

Due to increased domain sizes, each step in the extension of a local search on a SCMP takes more work than on SAT problems. In SCMP, the number of variable-value pairs to be considered in an unsatisfied clause may be very large. Therefore, each step requires more computation. For this reason, the search provides for limiting the number of variable-value pairs that are evaluated before each move is made. One way to limit the number of variable-value pairs to be evaluated in each step is to evaluate only a fixed number of randomly selected values for every variable in the clause.

3.2 Neighborhood structures and encoding schemes

Local search algorithms search for good solutions in a space of legal states or solutions. A neighborhood of local search defines the neighbors of a state that a local search can reach from that state in a single move. For a local search such as WalkSAT that changes the value of only one variable at each step, the neighboring states differ in exactly one variable's assignment.

An encoding scheme defines a unique local search neighborhood. We now discuss the neighborhood structure of the three different encoding schemes we described in Section 2.

The neighborhood structures of unary encoding is not suitable for WalkSAT and its variants that change the assignment of one variable at each step. To see this, recall that each value of a variable is represented by a Boolean variable. Consider a Boolean variable x that represents the value d of a variable X. When a local search chooses x and flips its value, it will enter an illegal state. If the value of x is flipped from T to F, the original variable X will not take value d in the next state, but indeed, variable X will not take any value in the new state at all. Similarly, if the value of x is flipped

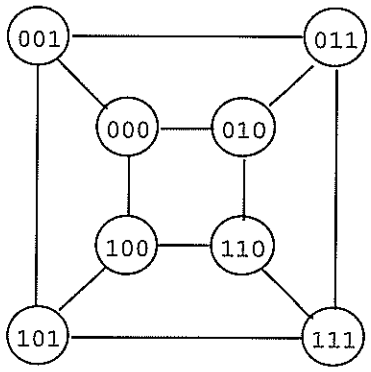

Figure 1: Neighborhood structure of three Boolean variables.

from F to T, the original variable X will take two values, value d, and another value corresponding to the True value of a variable encoding another value of X.

In short, WalkSAT cannot be combined with unary encoding. In order to use unary encoding, substantial modifications to WalkSAT are required. One possible modification may allow more than one variable to be "flipped" at each step of a local search. Even multiple variables are allowed to be "flipped", it still requires special care to ensure that the next state is legal, i.e., each unencoded original variable has one and only one value in the new state. This will certainly introduce more computation to a local search, which may wipe out the saving of the encoding scheme. Another approach would allow WalkSAT to take states that violate hard constraints. When the variable domains are large, however, the overall search space can be extremely large.

Binary encoding introduces a special neighborhood structure for WalkSAT. Figure 1 shows the neighborhood structure of three Boolean variables, x_1, x_2 and x_3. Each node in this figure represents a legal state, and a state labeled as 000 means $x_3 = F, x_2 = F$ and $x_1 = F$, and state with 001 means $x_3 = F, x_2 = F$ and $x_1 = T$, and so forth. Two nodes are connected if the value in one node can be changed to the other by flipping one variable. For instance, node 100 is linked to node 101 because the states are reachable by a flip of x_1.

The imposed neighborhood structure of binary encoding can play two different roles in WalkSAT. Recall that only one Boolean variable is allowed to flip in each step of WalkSAT. On an unencoded SCMP, WalkSAT can change a selected variable to any value in its domain in a single step. In contrast, when the problem is binary encoded, WalkSAT may take as many as $\lceil \log_2 m \rceil$ steps to change the selected variable from one value to another value. Take domain $\{0, 1, 2, \cdots, 6, 7\}$ as example, whose binary neighborhood is shown in Figure 1. It takes 3 steps (or variable flips) to move from value 2 (node 010 in Figure 1) to value 5 (node 101) on a binary encoded problem, while it can move from 2 to 5 in a single step on an unencoded problem.

Therefore, binary encoding may force WalkSAT to make a long sequence of moves to search a small area in an unencoded problem.

Compared to binary encoding, k-encoding with a large k value provides more mobility to a local search algorithm, because there are $k-1$ alternative values for each variable with domain size k. This means that by change the value of one variable, WalkSAT can potentially move to $k-1$ neighbors, while in a binary encoding ($k=2$), there is only one neighbor corresponding to a particular variable that the algorithm can move to. However, compared to binary encoding, k-encoding uses more computation to find the appropriate next value of a selected variable.

Unencoded Variables provide larger neighborhoods than any encoding scheme as all possible values of a variable are accessible from one another. A local search such as WalkSAT would be expected to be most mobile when operating in such neighborhoods.

It should be highlighted that encoding schemes have at least two separate effects on a local search algorithm. One limits the number of neighboring variable assignments of original variables. This reduces the number of neighboring states to be evaluated in each step of a local search, and reduces the computation at each step.

The other effect of encoding methods is the fact that an encoding method can bring special, predetermined neighborhood structures to a local search. Such predetermined neighborhoods may have special effects on local search. As we will see in the next section, structured neighborhoods make WalkSAT more effective and efficient on some graph coloring problems.

It remains to be determined theoretically and experimentally whether k-encoding is beneficial to solving particular SCMP, and if so, what level of encoding is most effective.

4 Experimental Analysis

In this section we present a series of four experiments to explore and evaluate the effects of various encoding strategies on the effectiveness of the generalized WalkSAT algorithm. Before presenting these results, we first discuss the test problems and methods we used in test experiments.

4.1 Experiment setup

Our test problems are taken from over constrained graph coloring problems which are converted into general SCMP. In this conversion, all vertices in a graph coloring problem become variables, with domain size equal to the number of colors. Every edge of the graph is represented by a constraint for each color, specifying the penalty for it's two endpoints both having the same color. For unweighted graphs, constraints are equal weight for each edge, and the quality of a coloring is assessed by how few

adjacent vertices have the same color assignment. Similarly, weighted constraints are used to represent edges in weighted graphs. Colorings that violate high weight constraints are therefore evaluated more harshly than those that violate low weight constraints.

In all of the graph coloring problems considered, we provided fewer colors than the actual chromaticity of each graph in order to create over constrained optimization problems. It is therefore the goal of a search algorithm to find assignments which minimize the sum of the violated constraints.

Because k-encoding introduces additional hard constraints into problems where variable domains are not powers of k, we chose test problems with cardinality of colors equal to a power of k. While this limitation does not illuminate an important obstacle for k-encoding, it allows us to evaluate k-encoding in its purest form. Since we are interested in the effect of encoding schemes on local search performance, we intend not to introduce this side effect of the encoding. Specifically, we favored 16-coloring problems, as they allow comparison between 2-encoded (binary encoded), 4-encoded, and the unencoded versions of the problems.

We used graph coloring problem instances from a coloring problem archive [3], some of which are coloring problems converted from real applications. We also randomly generated coloring problem instances in our experiments.

Throughout this section we will feature the test results obtained from two problem instances from this archive. The first problem, le450_25a.col, is a Leighton graph with 450 vertices and 8260 edges (see [3] for detail). Because the graph has a known chromaticity of 25, we provide 16 colors to make the problem over constrained. The second problem, fpsol2.i.3.col, is a problem based on register allocation for variables in real codes (see [3] for detail). When cast as a graph, it has 425 vertices and 8688 edges, and we have allowed 16 colors, out of 30 required colors, for its optimal coloring. Throughout this section, we will refer to our featured problem instances simply as the *Leighton graph*, and the *Register problem*. In order to preserve the nature of these real-world problems as much as possible, the constraints are left unweighted, but are interpreted as soft constraints nonetheless.

In addition to the two featured problems, we have also run our experiments on several other problem instances, including 5-colorable and 15-colorable Leighton graphs (le450_5, le450_15* problem instances from the archive), and Register problems (mulsol.i.*, zeroin.i.*,inithx.i.* from the archive).

We experimented on dense and sparse random, unweighted and weighted graph coloring problems. The dense graphs have 75 vertices and 2,500 edges, and 120 vertices and 5,000 edges. We used 16 colors to color these dense graphs. The sparse graphs have 1,000 vertices and 8,000 edges using 4 colors. To create weighted graphs, the edges of the random graphs were assigned random weights between 1 and 100.

The results of the featured problems reported here are averaged over 30 independent searches of the WalkSAT algorithm under each experiment's conditions. Each search begins with a random legal variable assignment. The searches also restart

with a new random assignment anytime they make 500 moves on a plateau without making downward progress.

In our experiments, we observed that the performance of WalkSAT deteriorates if it make a random move with a fixed probability. The inclusion of such random moves not only delayed the algorithm's initial downward moves, but also prevented it from reaching solutions of reasonable quality in a long time. With a probability more than 10 percent, the quality of solutions that WalkSAT can reach flattens out after a certain amount of time, dampening any further search effort. With a probability more than one percent, WalkSAT is still impaired in its ability to make a quick descent as well as in its ability to reach good solutions in a long run. Therefore, in the following sections we will not report any results on WalkSAT using a probability of random moves.

In the next few sections, we will report the anytime performance of our extended WalkSAT algorithm on different encoded coloring problems and unencoded problems. The anytime performance of the algorithm measures the quality of solutions improving over the whole time spectrum during the algorithm's execution.

All the tests were run on an AMD Athlon Thunderbird 1GHz PC workstation, with 1.5GB memory, running RedHat Linux 7.0. The WalkSAT search was implemented in C/C++, and compiled with g++ version 2.96.

4.2 Comparison of encoding methods

The first experiment that we present is a comparison between the performance of generalized WalkSAT on unencoded SCMP and k-encoded SCMP. For each encoding scheme, we are interested in the highest quality of solution that the WalkSAT algorithm is able to find as time progresses. The quality of solution is measured as the sum of the weights of all soft constraints that are unsatisfied. Note that during a search the quality of the best solution is monotonically increasing, since the algorithm remembers the best solution found so far, even as it continues to explore lower quality assignments.

The plots in Figure 2 show the performance of WalkSAT on the Leighton graph and Register problem under different levels of k-encoding. Of the three encoding schemes, the unencoded problem facilitates the most effective search on the Leighton graph. Not only does the unencoded version outperform the k-encoded versions during the initial descent of the search - when almost every move is a successful downward move - but it also converges to higher quality solutions than either of the k-encoded versions. The 4-encoded input also definitively outperforms the 2-encoded input throughout the whole spectrum of the search.

All of the Leighton graph instances, as well as all random coloring problems that we have generated showed similar results as Figure 2(a). Each problem similarly shows k-encoding lagging far behind the unencoded versions, with highest degrees ($k = 2$) of encoding falling the furthest behind.

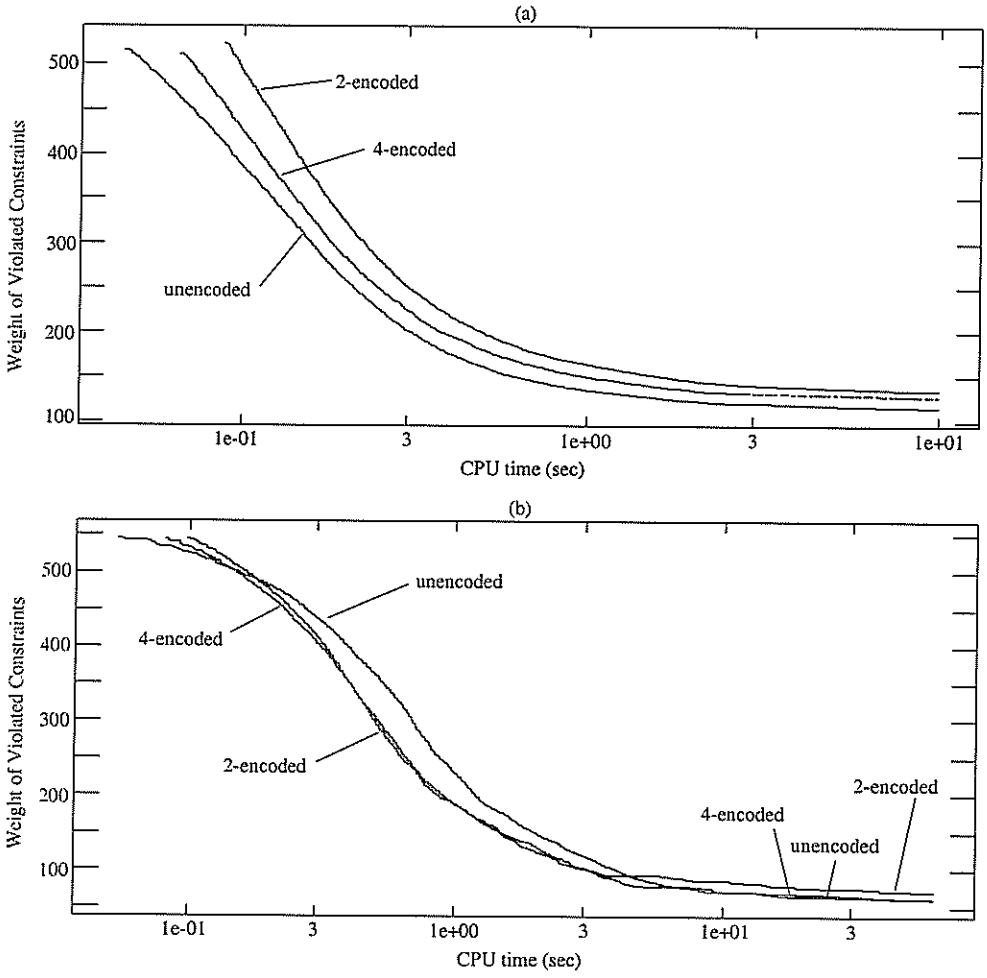

Figure 2: WalkSAT on k-encodings of the Leighton graph (a) and Register problem (b).

The Register problem, on the other hand, displayed a dramatically different result, as can be seen in Figure 2(b). On this problem, as well as all of the other Register scheduling problems that we tested, WalkSAT performance on k-encoded problem was far more competitive with the unencoded problems. In Figure 2(b), the 4-encoded problem performs far better than the unencoded version early in the search, and is remains very close to the quality of the encoded input late in the search.

One key observation that is evident in all of these experiments is that the k-encoded problems take longer to begin their search. This is because the search begins with a random assignment which requires each term of every clause to be compared to the assignment. Since the unencoded problem has fewer variables in its clauses than the k-encoded versions, it is able to evaluate the random assignments which are

generated both at the beginning of the search, and periodically when the search is stuck on a plateau.

Although creating and evaluating random assignments is quite slow for k-encoded problems, standard moves are made faster than in unencoded problems. Even including the slower random restart moves, WalkSAT still manages to make more moves on k-encoded problems. On the Register problem, WalkSAT is able to make 54 moves per second on the unencoded problem, while on the $k = 4$ and $k = 2$ encoded problems it makes 103 and 130 moves, respectively. The k-encoding had a much smaller effect on the speed of the search on the Leighton graph problem which allowed 619 moves/second on the unencoded version, and 758 and 767 moves/second on the 4- and 2-encoded problems. These results are not unexpected: As explained in Section 3.1, the time WalkSAT takes to make one move is approximately proportional to the number of variables in the clause plus the number of neighboring assignments to be evaluated, and the overhead costs associated with making the move. Still, the time it takes to evaluate a value change in one of the variables depends most significantly on how many constraints the variable is involved in.

Despite making faster moves on both problem instances, the cause of the mixed success of k-encoding is not immediately clear. There are two important ways in which k-encoding affects the way WalkSAT performs on SCMP. First, because encoded variables have smaller neighborhoods - that is, a smaller number of values that can be reached by changing a singe variable - than their unencoded counterparts, the k-encoded problems do not allow WalkSAT to consider as many neighboring assignments when making each move. The effects of this difference are studied in experiments reported in Section 4.4. The second effect of k-encoding is that the neighborhoods of the variables in a clause is rigidly structures and dependent on the current assignment of the variables. The effects of this are studied in experiments reported in Section 4.5.

Although the 4-encoded version of the Register problem gave results that were very competitive with those of the unencoded problem, the unencoded problem did give the best results in all long-running searches that we tried.

4.3 Effects of limited neighborhood size

Our generalized WalkSAT evaluates the possibility of changing each variable of a clause to each of the values in their domain, save their current values, before making each move. As a result, unencoded and k-encodings with large values of k allow WalkSAT to check more neighbors in each step than their heavily-encoded counterparts. Because the speed of a move is almost proportional to the number of neighbors that are evaluated, plus some overhead, the unencoded problems make WalkSAT move much more slowly than their k-encoded counterparts.

We isolate the effects of the number of neighboring assignments that are evaluated at each move by enforcing limits on the number of assignments that are considered in

a single move of an unencoded problem. In both of our featured problem instances, we have 16 possible colors for each variable. Because each constraint involves two variables, and each variable has 15 alternate values which can be evaluated, WalkSAT normally will evaluate 30 neighboring assignments before making each move.

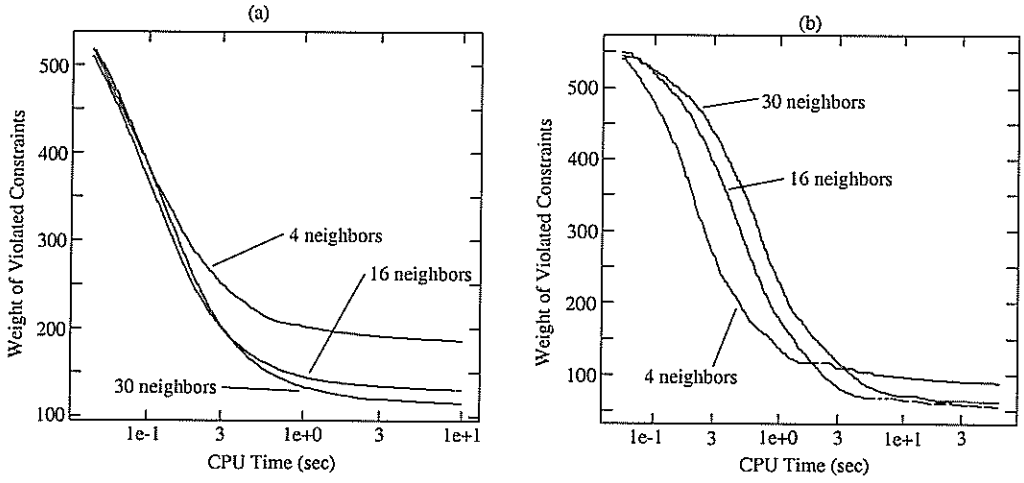

Figure 3: WalkSAT on limited neighborhood structures on Leighton graph (a) and Register problem (b).

This experiment explores the effects of limiting the number of neighboring states which are allowed to be evaluated. Rather than checking all 15 alternate values of each variable in an unsatisfied clause, we evaluate a randomly selected subset of each variable's alternate values. Figure 3 shows the results of evaluating different sized subsets of each variable's domain in our featured problems.

It is clear that in these two problems, as well as other problems that we experimented on, overly limiting the number of neighbors is harmful to the long term effectiveness of WalkSAT, the eventual outcome of the search. Early in the search, however, it can be seen that faster moving searches, which are checking only a subset of their neighboring states at each move, are able to descend to better solutions quickly. The search limited to four neighbors per move is in fact the most effective for the first two seconds in the Register problem. Although this early timeframe may be insignificant when considering the effectiveness of a search on SCMPs of this size and nature, it is reasonable to say that other SCMPs may be best served with faster moving searches. For example, in an extraordinarily large problem, or if time is limited, a fast moving search which limits its moves could significantly outperform a search which checks all of it's neighbors at each move.

4.4 Structural effects of k-encoding

In addition to checking fewer neighboring assignments during each move, WalkSAT on k-encoded problems also evaluates a particular subset of the neighbors that an unencoded problem in the same assignment would evaluate. One such example can be seen in the cube in Figure 1 on page 8: while and unencoded variable with domain $[0 \sim 7]$ assigned to 0 can be changed to 7 other values, its 2-encoded variable set $x_0 = 0, x_1 = 0, x_2 = 0$ can only be flipped to reach 3 particular values: 1, 2, and 4.

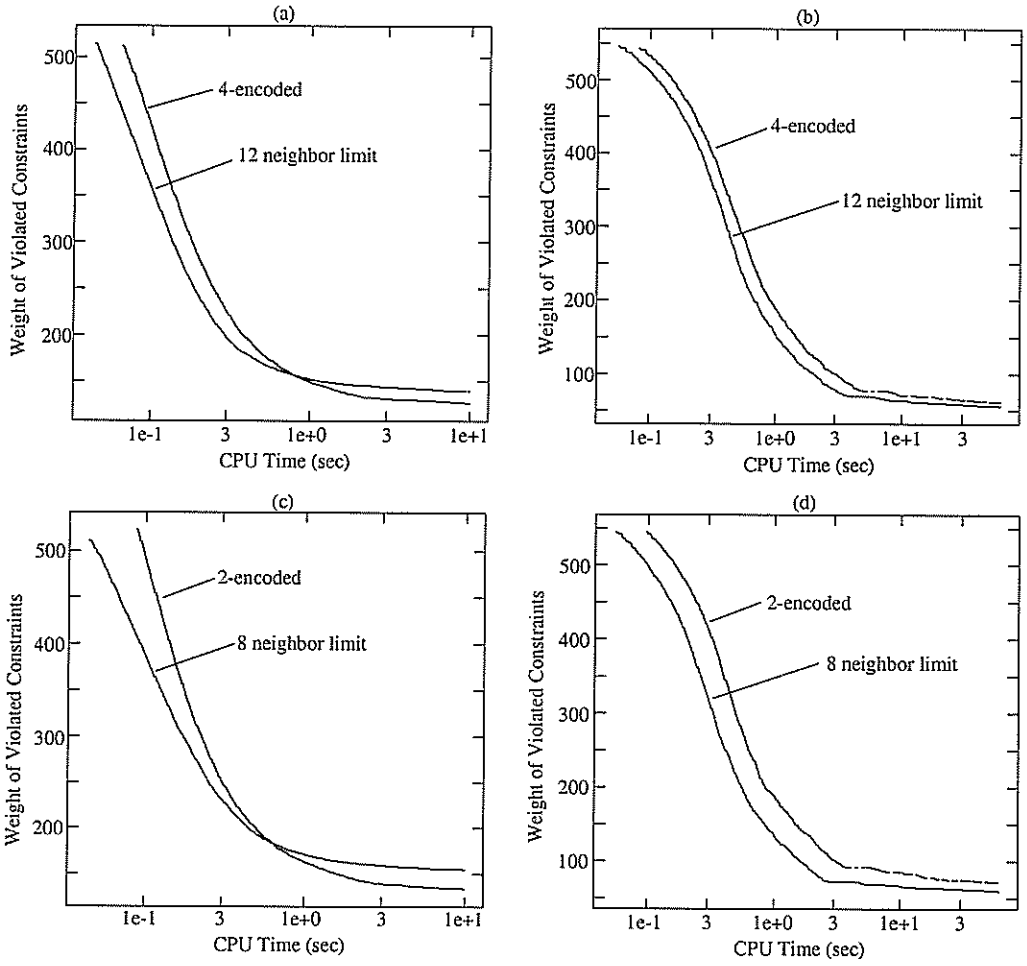

Figure 4: WalkSAT on limited neighborhood structure vs. k-encoding structure for the Leighton graph (a)(c) and Register problem (b)(d).

We would like to assess the effects of this rigid neighborhood structuring incurred in k-encoding independently of the fact that k-encoding limits the number of neighbors that WalkSAT evaluates at each move. To do this we compare WalkSAT's performance on k-encoded problems with their unencoded counterparts limiting the

number of neighbors evaluated during each move to the same number that the k-encoded problem will evaluate. In Figures 4(a) and 4(b), our 16 color Leighton graph and Register problem are used to compare the performance of WalkSAT on a 4-encoding vs. unencoded input limited to evaluating 6 assignments per variable in the unsatisfied clause. Figures 4(c) and 4(d) uses the same problems encoded with $k = 2$ and unencoded with a limit of four values per variable in the unsatisfied clause.

The results of this experiment are very different on each of the two problems in the figures. The Leighton graph shows the k-encoding as the clear winner, allowing WalkSAT to find higher quality solutions, regardless of the search time. All of the Leighton graph problems, as well as all of the randomly generated weighted and unweighted graphs that we tested showed this result.

As shown in Figures 4(b) and 4(d), on the other hand, WalkSAT on the Register problem performs much better when neighbor-checking is randomly limited, rather than when the problem itself is k-encoded. A similar result was found in all of the Register Scheduling problems tested.

In both problem instances, WalkSAT moves slightly faster on the unencoded problem which is limited to a subset of its neighbors than on the k-encoded problems. As discussed in section 3.1, this is because in addition to evaluating the clauses for each value that is changed, the algorithm must evaluate the current assignment of the variables. In the experiment involving 2-encoding, the 8 variables in a clause must be evaluated at both their current values and their one alternate values, meaning that the affected clauses must be evaluated 16 times in order to choose among the 8 neighboring states. When the unencoded problem is limited to 8 neighbors to check in the clause, it needs only evaluate the clauses 10 times: twice for the two variables current values, and once for each of the 8 neighboring assignments. (need work)

The difference in movement speeds is far too small to account for the differences in quality of solution found late in the search by each strategy. The rigid neighborhood structure of the variables created by k-encoding has a strong positive effect on the performance of WalkSAT on the Leighton graph and randomly generated graphs we have tried, but an equally strong negative effect on the Register scheduling problems. It is likely that structuring in the constraints of the different problems have different interactions with the structure limiting the way the variables introduced by k-encoding can be changed.

It is perhaps expected that the randomly limited search would find better solutions that a search whose possible moves are limited to a rigid structure. If both searches were stuck in a local minimum, for example, it seems that the search which can consider many different and random subsets of 30 moves on a single clause will have more chances to escape than a search which must choose for among a similar subset of moves each round.

The behavior exhibited by the Leighton graph and random graph coloring problems was therefore largely unexpected. One explanation of this phenomenon is that certain features of the search topology in these problems may trap searches with

randomly limited movement options in local minimums while searches of k-encoded problems are able to escape. Such a trap could be one where a single large upward move, denoted as M, is the only escape. Since our general WalkSAT always takes the least uphill move, a k-encoded problem could escape the minimum if all of its other moves in the encoded neighborhood move further uphill than move M. A search which considers only a random subset of its moves may be very unlikely to take the escape move M if there are many less-uphill moves available to it.

From this experiment we can conclude that the neighborhood structure introduced by k-encoding may be helpful to a WalkSAT search in some SCMPs. We observe that the structure is beneficial to the search of Leighton graph coloring, as well as random graph coloring problems, but harmful to Register scheduling problems. We do not know what differences in the structure of these problems cause the differences in search performance now, but this will be a topic of future work.

As observed in the Register problem, it is beneficial in some application domains to restrict the number of neighboring states to be evaluated in each move. The Leighton graph and random graphs also show that k-encoding can outperform randomly limited searches on certain problem types. The possibility of problems that can benefit from both restricted and rigidly structured neighbor sets motivates us to further study k-encoding to speed up local search algorithms.

4.5 Simulating k-encoding with WalkSAT

We developed a simulated approach to k-encoding, which we call *simulated k-encoding*. A local search using simulated k-encoding moves exactly as it would if a problem were actually k-encoded. Simulated k-encoding for generalized WalkSAT works as follows: read-in an unencoded SCMP without restrictions on variable domains; for each move, select an unsatisfied clause as normal; for each variable of the clause, consider its current assignment, and only consider changing its value that would be possible in a k-encoded problem.

For each variable, a table can be created to store the neighboring values of a variable value in a k-encoding. Such a table would essentially store the structure of a k-encoded variable similar to the one shown in figure 1. We call such a table a *k-neighboring table*. A k-neighboring table allows the search to immediately determine the subset of neighboring assignments to evaluate each time it makes a move. The k-neighboring tables of all variables of a problem can be initially compiled before the search starts.

Figure 5 shows the performance improvement of WalkSAT using simulated 2-encoding with respect to WalkSAT using original 2-encoding on the Leighton graph. Unlike in all other graphs shown earlier, each plotline in Figure 5 is averaged from searches that were run for exactly 10,000 moves. These searches were also not allowed to restart with a random assignment. With the logarithmic time axis of the graph, the simulated k-encoding can be seen to be descending faster than true k-encoding

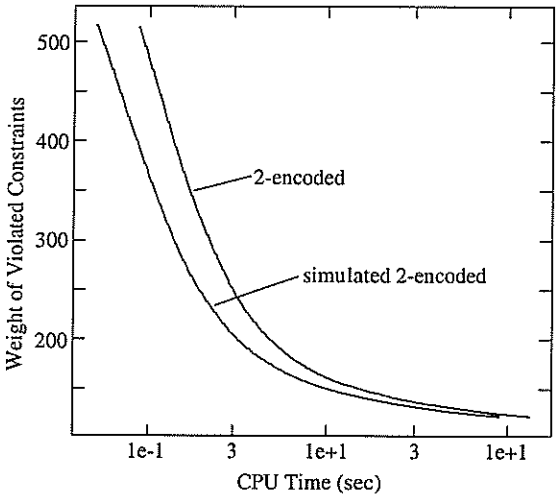

Figure 5: Speedup using simulated k-encoding on the Leighton graph.

by a significant constant factor. Also note that the simulated k-encoding finished its 10,000 moves 12% faster than the true k-encoded problem.

There are a number of advantages to simulated k-encoding over true k-encoding:

1. The memory needed to represent the problem is more compact as there are fewer variables, and shorter clauses involving fewer variables. The savings in memory are more substantial for SCMP with variables that have large domains.

2. There are several options available to avoid introducing hard constraints to the problem as a result of the simulated k-encoding. For example, a variable X with domain size m could be k-encoded even if m is not a power of k, by simply simulating k-encoding with $\lfloor \log_k m \rfloor$ variables, but removing any forbidden values by not including them in the k-neighboring table for X.

3. The generalized WalkSAT algorithm can run fast on simulated k-encoded problems because it needs to evaluate the constraints fewer times per move. As described in section 3.1, to make a single move WalkSAT must evaluate the constraints for every variable's current variable, as well as for every neighboring assignment that is considered. Since a k-encoded problem will have more variables in the unsatisfied constraints, there are more constraints to be evaluated than in the encoded problem.

If a search algorithm can benefit from k-encoding of a particular CSP, the same benefits should be attainable with greater efficiency by modifying the search to simulate k-encoding.

5 Related Work and Discussions

Soft constraints were initially proposed and studied under the notions of fuzzy constraints [12], uncertain constraints [5], soft constraints [1, 2] and valued constraints [13, 14]. In these studies, special structures have been imposed onto the values of constraints. For instance, in the soft constraint framework [1, 2], semiring structures were considered, and in the valued constraint framework [14], special valuation systems were used. The main purpose of introducing special structures was to study the representational power of soft CSPs and the degree of consistency that can be achieved by various constraint analysis processes.

In this research, we do not restrict ourselves to soft constraints with a structure, since the constraints in a real-world SCSP may not necessarily be well structured or have well understood structure. In this paper, we consider general problem-solving methods for SCSP, local search methods in particular.

We study over constrained SCMPs in particular, where there exists no value assignment to satisfy all constraints. Over constrained CSP was first studied under the notion of partial constraint satisfaction [6], which concerned finding a solution to satisfy a subset of a CSP when it is over constrained. Over constrained CSP was also studied under the notion of maximum CSP (MAX-CSP) [17], which concerned finding a solution to minimize the total number of violated constraints. Depth-first branch-and-bound was used to solve MAX-CSP, and its performance was studied through experiments [17].

The previous work on MAX-CSP is also related to our research because both consider optimization problems. There are two differences, however. The first is that our computational model, soft constraint minimization problem (SCMP), can be considered as an extension of MAX-CSP where constraints may have different weights. The second is research objectives. In our research, we are interested in approximate solutions as well as optimal solutions, whereas in the previous work on MAX-CSP the main goal is to find optimal solutions. This difference has been reflected by the different approaches to finding solutions. We applied local search, while the previous MAX-CSP work used depth-first branch-and-bound.

Inspired by an initial success of applying local search algorithms for SAT to planning problems [10], much attention has been attracted to apply and extend local search algorithms for SAT to CSP [4, 7, 9]. One approach considered was SAT encoding, which represents CSP with Boolean variables.

Two SAT encoding schemes have been studied, but under different terminologies. The encoding that uses a new propositional variable for each value of a CSP variable has been called unary transform [7], sparse encoding [9], and direct encoding [18]. The encoding that represents a CSP variable with a set of Boolean variables in a binary form has been called binary transform [7], compact encoding [9], and log encoding [18]. In this paper, we refer to these two encoding methods as unary encoding and binary encoding, respectively.

One of the major contributions of this paper is the k-encoding scheme, proposed in Section 2.2. It extends the concept of SAT encoding and take binary encoding as a special case. This general encoding method helped us to better understand encoding schemes in general, as well as the performance of local search algorithms.

Another main issue regarding applying local search for SAT to CSP was the performance of a local search algorithm on CSP. The performance of efficient local search algorithms for SAT, such as WalkSAT and its variants, have been studied on a few selected CSPs, including planning [4], graph coloring [7] and binary CSP and Hamilton Circuit problems [9]. Experimental results from previous studies have generally shown that binary encoding is less effective than unary encoding.

As we have discussed in Section 3.2, WalkSAT cannot be directly applied to an unary encoded problem. Since one of our objectives of our current research is to apply WalkSAT to SCMP, we did not experiments on unary encoding. We leave it to our future work.

Another difference between our research and the previous work is that we investigated the anytime performance of an extended WalkSAT algorithm on SCMPs, which are optimization problems. We have not only examined the effectiveness of WalkSAT using various encodings, i.e., the quality of solutions that the algorithm can reach with sufficient time, but also the quality of solutions that the algorithm can find at all times during its execution. Our anytime results can in fact be used to explain some of the previous experimental results on CSP. For example, it was observed that local search using binary encoding did not find solutions to colorable graph coloring problems [7]. Our anytime results of extended WalkSAT showed that binary encoding is not very effective, and a local search algorithm using this encoding method may never find the satisfying assignment to the problems.

6 Conclusions

In this research, we have considered the application and extension of local search, and the WalkSAT algorithm in particular, to soft constraint optimization problems. We used a computational model of soft constraint optimization problems, which we call soft constraint minimization problems (SCMP). We studied the existing encoding schemes, unary and binary encodings, and proposed a new, general scheme called k-encoding. We analyzed anytime performance of WalkSAT and effects of different encoding schemes on finding approximate solutions of SCMP.

Based on our experiments on many graph coloring problems, we come to the following conclusions.

- WalkSAT performs better on all of our test problems when they are not encoded, provided enough execution time. On certain problems we tested, WalkSAT using k-encoding with large k is competitive. Comparing levels of encoding,

k-encoded problems were more difficult to search by lower values of k. Binary encoded ($k = 2$) problems are the most difficult.

- On all of our unencoded test problems, limiting the number of neighboring assignments evaluated during each move always proved harmful. While limiting these evaluations does allow the search to make moves faster, and sometimes find higher quality solutions sooner than the slower moving searches, such limitations generally make WalkSAT less effective in the long run.

- The neighborhood structure of k-encoding can be simulated by a search algorithm on an unencoded SCMP, and such a simulation is significantly faster than a search on k-encoded input.

- On many problem instances, WalkSAT was able to find better solutions when it was k-encoded and bound to the resulting neighborhood structure. Although k-encoding's contribution of structure is generally overwhelmed by the fact that its neighborhood size is greatly reduced, some SCMP problems may still benefit. Several ideas for improving the performance of k-encoding are presented in the next section.

7 Future Work

Our current research can be extended in many directions. In our future work, we plan to investigate the following issues:

- *Increasing the neighborhood size in WalkSAT.* Based on our experiments, increasing the number of neighbors considered at each move improves the performance of WalkSAT. One possible extension to WalkSAT is to allow the search to consider changing the variables in more than one unsatisfied constraint.

- *The scope of k-encoding's benefits.* For some problems WalkSAT using k-encoding seems to find better solutions than WalkSAT randomly selecting a restricted number of neighbors at each move. This may be because k-encoding has an effect of escaping local minima. It would be interesting to test if the benefits of k-encoding are reduced or negated when explored by searches employing additional strategies for escaping local minima, such as simulated annealing, tabu lists, or novelty search.

- *Taking full advantage of k-encoding structures.* When problems are encoded, many semantics get lost. However, k-encoding can help to preserve the semantics. The local search neighborhood structure of a k-encoding can be exploited to represent some natural embedded structure among the variable values. For example, if a variable represents types of shirts, it may be desirable to make all

of the red shirts close neighbors in the encoded neighborhood. This way, the search will be forced to explore semantically similar assignments. The neighborhood structure may also be used to bring semantically distant values into close proximity to increase the mobility of a search.

References

[1] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of ACM*, 44(2):201–236, 1997.

[2] P. Codognet and F. Rossi. Notes for the ECAI2000 tutorial on Solving and Programming with Soft Constraints: Theory and Practice. available at http://www.math.unipd.it/ frossi/papers.html.

[3] Graph coloring problem instance archive. http://mat.gsia.cmu.edu/color/instances.html.

[4] M. D. Ernst, T. D. Millstein, and D. S. Weld. Automatic SAT-compilation of planning problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence, (IJCAI-97)*, pages 1169–1176, nagoya, Japan, 1997.

[5] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: A probabilistic approach. In *Proc. European Conf. on Symbolic and Qualitative Approaches to Reasoning and Uncertainty*, pages 97–104, 1993.

[6] E. C. Freuder and R. J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[7] A. M. Frisch and T. J. Peugniez. Solving non-Boolean satisfiability problems with stochastic local search: A comparison of encodings. available from http://www.cs.york.ac.uk/aig/publications.html.

[8] J. Gu. Efficient local search for very large-scale satisfiability problems. *ACM SIGART Bulletin*, 3(1):8–12, 1992.

[9] H. H. Hoos. Sat-encodings, search space structure, and local search performance. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence, (IJCAI-99)*, 1999.

[10] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, 1996.

[11] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 321–326, 1997.

[12] A. Rosenfeld, R. A. Hummel, and S. W. Zucker. Scene labelling by relaxation operations. *IEEE Trans. on Systems, Man, and Cybernetics*, 6(6):420–433, 1976.

[13] T. Schiex. Valued constraint networks. In *Proceedings of the 6th Conference on Principles and Practice of Constraint Programming (CP-00)*, 2000.

[14] T. Schiex, H. Fargier, and G. Verfaille. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, (IJCAI-95)*, pages 631–637, 1995.

[15] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343, Seattle, WA, 1994.

[16] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, San Jose, CA, July 1992.

[17] R. J. Wallace. Enhancements of branch and bound methods for the maximal constraint satisfaction problem. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 188–195, 1996.

[18] T. Walsh. Sat v csp. In *Proceedings of the 6th Conference on Principles and Practice of Constraint Programming (CP-00)*, 2000.