Constraints, 7, 151–165, 2002

A New Approach for Weighted Constraint Satisfaction

HOONG CHUIN LAU lauhc@comp.nus.edu.sg
School of Computing, National University of Singapore, 3 Science Drive 2, Singapore 117543

Abstract. We consider the Weighted Constraint Satisfaction Problem which is an important problem in Artificial Intelligence. Given a set of variables, their domains and a set of constraints between variables, our goal is to obtain an assignment of the variables to domain values such that the weighted sum of satisfied constraints is maximized. In this paper, we present a new approach based on randomized rounding of semidefinite programming relaxation. Besides having provable worst-case bounds for domain sizes 2 and 3, our algorithm is simple and efficient in practice, and produces better solutions than some other polynomial-time algorithms such as greedy and randomized local search.

Keywords: constraint satisfaction, randomization, semidefinite programming, approximation algorithm

1. Introduction

An instance of the binary Weighted Constraint Satisfaction Problem (W-CSP) is defined by a set of variables, their associated domains of values and a set of binary constraints governing the assignment of variables to values. Each constraint is associated with a positive integer weight. The output is an assignment which maximizes the weighted sum of satisfied constraints. W-CSP is a generalization of important combinatorial optimization problems such as the Maximum Cut Problem. Many real-world problems such as scheduling and time-tabling can also be represented as W-CSP. In scheduling for example, our task is to assign resources to jobs under a set of constraints, some of which are more important than others. Most often, instances are over-constrained and no solution exists that satisfies all constraints. Thus, our goal is to find an assignment which maximizes the weights of the satisfied constraints.

The purpose of this paper is to describe a novel approach based on semidefinite programming. This approach combines the strengths of complete algorithms in terms of guaranteeing optimality (albeit fractionally) and incomplete algorithms in terms of fast run times. In line with the rapid increase in the speed of computing and the growing need for real-time efficient scheduling, it makes sense to explore ways of obtaining provably better schedules at some extra computational cost, short of going all the way towards the usually futile attempt of finding a guaranteed optimal schedule.

The highlight of our approach is that the solution obtained has a provable worst-case bound in terms of weight when compared with the optimal solution. This contrasts with conventional incomplete algorithms which are empirically good but are not guaranteed to perform well in the worst case. In this paper, we establish the worst-case bounds for the class of W-CSP problems with domain sizes 2 and 3. Several interesting NP-hard

optimization problems fall within this class, including graph 3-coloring, maximum cut and maximum 2-satisfiability.

Besides being theoretically interesting, the knowledge of the worst-case performance yields at least three benefits:

1. It gives us some peace of mind that the algorithm will never perform embarrassingly poorly in the worst case. This assurance is an important consideration, especially in mission-critical applications.
2. It provides a priori bounds on the optimal solution in polynomial time. This has implications on problem solving. If we have a complete algorithm to solve our problem, our approach computes an initial solution which provides a lower bound on the optimal solution value. On the contrary, if we have an incomplete algorithm, our approach yields a provable upper bound, thereby allowing us to measure the quality of solutions produced by the incomplete algorithm without having to compute the optimal solution.
3. In over-constrained systems, the upper bound helps the user to determine which constraints should be relaxed so that the instance becomes solvable.

1.1. Related Works

Finding optimal solutions of W-CSP is computationally hard, since W-CSP generalizes NP-hard optimization problems such as the Maximum Cut Problem. In the CSP research community, work in W-CSP is not as abundant as work in the standard CSP. Freuder and Wallace [3, 4] gave the first formal definition of PCSP which is a special case of W-CSP having unit weights. For PCSP, the objective is to satisfy as many constraints as possible. Freuder and Wallace proposed a polynomial time algorithm based on reverse breadth-first search to solve PCSP whose underlying constraint network is a tree. For the general PCSP, they proposed a general framework based on branch-and-bound and its enhancements in [18, 20]. Larrosa and Meseguer [9] proposed heuristics which can be incorporated into a forward checking algorithm. Incomplete algorithms which yield near-optimal solutions have also been investigated. These include heuristic repair methods [21], the connectionist architecture GENET [14] and guided local search [16]. More recently, more efficient heuristic methods have been proposed by Wallace [19] and Galinier and Hao [6].

In [10], Lau performed a worst-case analysis of local search for PCSP. In [11], Lau and Watanabe obtained theoretical results for finding approximate solutions for W-CSP. They proved that even computing a solution to within a constant factor of the optimal is hard. In this paper, our main purpose is to introduce a new approach for solving W-CSP based on the work in [11]. We will give a careful account of the mathematical modeling, the algorithm design, as well as experimental performance of our approach. The experimental results turn out, to our pleasant surprise, to be much stronger than the theoretical worst-case bound.

1.2. Randomized Rounding

Our approach is heavily based on the notion of *randomized rounding*. Randomized algorithms have proved to be powerful in the design of approximate algorithms for combinatorial optimization problems. An interesting and efficient algorithmic paradigm is that of randomized rounding, due to Raghavan and Thompson [13]. The key idea is to formulate a given optimization problem as an integer program and then find an approximate solution by solving a polynomial-time solvable convex mathematical program such as a linear program. The linear program must constitute a "relaxation" of the problem under consideration, i.e., all integer solutions are feasible for the linear program and have the same value as they do in the integer program. One easy way to do this is to drop the integrality conditions on the variables. Given the optimal fractional solution of the linear program, the question is how to find a good integer solution. Traditionally, one rounds the variables to the nearest integers. Randomized rounding is a technique which treats the values of the fractional solution as a probability distribution and obtains an integer solution using this distribution. Raghavan and Thompson showed, using basic probability theory, that the values chosen under the distribution do in fact yield a solution near the expectation, thus giving good approximate solutions to the integer program.

1.3. Semidefinite Programming

Essentially, we will solve W-CSP using what is known as *randomized rounding of a semidefinite program relaxation*. A semidefinite program is the optimization problem of a linear function of a symmetric matrix of variables subject to linear equality constraints and the constraint that the matrix be positive semidefinite. An $n \times n$ matrix F is said to be positive semidefinite if and only if $x^T Fx \geq 0$ for all real n-dimensional vectors x. Semidefinite programming is a generalization of linear programming and a special case of convex programming. Using interior point methods, one can show that semidefinite programming is solvable in polynomial time under some realistic assumptions (see [1]). Semidefinite programming, like linear programming, has been an active research topic in the Operations Research community; for details, see a good survey written by Alizadeh [1]. In practice, there are solvers which yield solutions quickly for reasonably large instances, as our experiments would show.

Our idea is to represent W-CSP as a quadratic integer program and solve a corresponding instance of semidefinite programming. The solution returned by the semidefinite program is then rounded to a valid assignment by randomized rounding. This approach yields a randomized algorithm. To convert it into a deterministic algorithm, we apply the *method of conditional probabilities* which is a well-known probabilistic method in combinatorics. The interested reader is referred to Alon and Spencer [2] for details.

Recently, Goemans and Williamson apply a similar approach to find approximate solutions for the Maximum Cut Problem [7]. Besides having a strong provable worst-case bound, their computational experiments show that, on a number of different types of random graphs, their algorithm yields solutions which are usually within 4% from the optimal solution.

Our approach offers a *polynomial-time algorithm* which can be efficiently implemented. Our experiments illustrate that this approach can solve, in real-time, problems whose sizes are beyond what enumerative search algorithms can handle. It is thus a candidate for tackling large-scale optimization problems dynamically. More precisely, we illustrate experimentally that, for random satisfiable problems with number of variables and domain sizes $(64, 2)$ and $(20, 5)$, this approach yields solutions within 4% from the optimal, using less than 1 minute CPU time on a SUN Sparc 10 station. This is significantly better than the other polynomial-time algorithms under comparison.

1.4. Organization of Paper

This paper is organized as follows. Section 2 gives the definitions and notations which will be used throughout the paper. In Section 3, we introduce the method of conditional probabilities and derive a linear-time greedy algorithm which can be used to derandomize our randomized rounding algorithms. We prove that a naive application of the greedy algorithm always returns a solution whose weight is guaranteed to be a fraction of s times the total weight, where $0 \leq s \leq 1$ is the *strength*[1] of the constraints. In Section 4, we show how to formulate W-CSP by quadratic integer programs. We discuss how randomized rounding can be used to yield solutions which have a constant worst-case bound for domain size $k \leq 3$. In Section 5, we compare the performance of our algorithm with other approaches experimentally on random W-CSP instances.

2. Preliminaries

Let $V = \{1, \ldots, n\}$ be a set of variables. Each variable has a *domain* which contains the set of values that can be assigned. For simplicity, we assume that all domains have fixed size k and are equal to the set $K = \{1, \ldots, k\}$. A *constraint* between two variables i and l is a binary relation R over $K \times K$ which defines the pairs of values that i and l can take simultaneously. Let $\#R$ denote the number of such pairs of values. Given an assignment $\sigma : V \longrightarrow K$, the constraint is said to be *satisfied* iff the pair (σ_i, σ_l) is an element of the relation. The Weighted Constraint Satisfaction Problem (W-CSP) is defined by a set V of variables, a collection M of constraints, integer k, and a weight function $w : M \longrightarrow Z^+$. The output is an assignment $\sigma : V \longrightarrow K$ such that the weighted sum of satisfied constraints (or simply weight) is maximized.

Denote by W-CSP(k) the class of instances with domain size k. For each constraint $j \in M$, let w_j, R_j and $s_j = \#R_j/k^2$ denote its weight, relation and *strength* respectively; let α_j and β_j denote the indices of the two variables incident on constraint j; and let $c_j(u, v) = 1$ if $(u, v) \in R_j$ and 0 otherwise. That is, $c_j(u, v)$ indicates whether the value pair (u, v) is consistent in constraint j. Let $s = \sum_{j \in M} w_j s_j / \sum_{j \in M} w_j$ denote the strength of a W-CSP instance (i.e., the weighted average strengths of all its constraints). Note that $s \geq 1/k^2$ because a constraint relation contains at least 1 out of the k^2 possible pairs. A W-CSP instance is *satisfiable* iff there exists an assignment which satisfies all constraints simultaneously.

In this paper, we will formulate a W-CSP instance as a quadratic integer program (QIP). A QIP has the form:

$$\begin{aligned} \text{maximize} \quad & \sum_{i,l} a_{il} x_i x_l + \sum_i b_i x_i \\ \text{subject to} \quad & \sum_i e_{ij} x_i = d_j \quad \text{for all constraints } j \\ & x_i \text{ integer}, \quad \text{for all variables } x_i. \end{aligned}$$

where x_i's are the decision variables and a_{il}, b_i, e_{ij}, and d_j are coefficients.

We say that a maximization problem P *can be approximated within* $0 < c \leq 1$ iff there exists a polynomial-time algorithm A such that for all input instances y of P, the ratio $A(y)/OPT(y)$ is at least c, where $A(y)$ and $OPT(y)$ denote the objective value of the solution returned by A and the optimal objective value of y respectively. The quantity c is commonly known as the *performance guarantee* or *approximation ratio* for P. The ratio is *absolute* if the denominator is the maximum possible objective value instead of $OPT(y)$. In the case of W-CSP for example, the maximum possible objective value is the sum of edge weights, although the optimal value can be much smaller. This means that if we can prove an absolute ratio of c, the performance guarantee is at least c. Observe that the ratio is as close to 1 as the solution is close to an optimum solution.

3. Method of Conditional Probabilities

In this section, we introduce the method of conditional probabilities. We derive an efficient linear-time greedy algorithm which can be used to derandomize the randomized rounding algorithm presented later. We will also analyze the worst-case performance of a naive application of the greedy algorithm.

Consider an instance of W-CSP. Suppose we are given an n by k matrix $\Pi = (p_{iu})$ such that all $p_{i,u} \in [0..1]$ and $\sum_{u=1}^{k} p_{i,u} = 1$ for all $1 \leq i \leq n$. If we assign each variable i independently to value u with probability $p_{i,u}$, we obtain an assignment whose expected weight is given by:

$$\overline{W} = \sum_{j \in M} w_j \times \Pr[\text{constraint } j \text{ is satisfied}] = \sum_{j \in M} w_j \left(\sum_{u,v \in K} p_{\alpha_j,u} \cdot p_{\beta_j,v} \cdot c_j(u,v) \right).$$

The method of conditional probabilities specifies that there must exist an assignment whose weight is at least $\overline{W}$ and that such an assignment can be found deterministically in polynomial time provided that certain conditional probabilities can be computed efficiently.

We can construct a complete assignment of expected weight at least $\overline{W}$ by assigning the variables incrementally via a greedy algorithm described as follows. Suppose we have a partial assignment with some unassigned variables. If these variables are to be assigned according to the distribution Π, then the expected weight of the resulting assignment can be computed. Let $\widetilde{W}$ be the state variable which stores this expected weight during the

greedy algorithm. The greedy algorithm is simply to iteratively set an unassigned variable to the value such that $\widetilde{W}$ is *maximized* over all possible values assignable to that variable. The greedy (derandomization) algorithm is coded as follows:

procedure *Greedy*:
  set $\widetilde{W} = \overline{W}$;
  for all $i = 1, \ldots, n$ **do**
    for all $u \in K$ compute $\widetilde{W}_u$; $(*)$
    assign i to v s.t. $\widetilde{W}_v$ is maximized;
    set $\widetilde{W} = \widetilde{W}_v$;
  endfor

In the above algorithm, $\widetilde{W}$ is initialized to $\overline{W}$, since no variables have been assigned yet. At the beginning of iteration i, $\widetilde{W}$ is the expected weight of the partial assignment where variables $1, \ldots, i-1$ are fixed and variables $i, \ldots, n$ are assigned according to the distribution Π. $\widetilde{W}_u$ denotes the expected weight of the same partial assignment, except that variable i is fixed to the value u. From the law of conditional probabilities:

$$\widetilde{W} = \sum_{u=1}^{k} \widetilde{W}_u \cdot p_{i,u}.$$

Since we always pick v such that $\widetilde{W}_v$ is maximized, $\widetilde{W}$ is nondecreasing in all iterations. Hence, at the end of the greedy procedure, we will have obtained a complete assignment whose weight is at least the expected weight $\overline{W}$.

Algorithmically, Step $(*)$ of *Greedy* can be implemented as follows. By conditional probability, $\widetilde{W}_u$ is just the current $\widetilde{W}$ offset by the change in probabilities of satisfiability of those constraints incident to variable i, conditioned by assigning i to value u. More precisely,

$$\widetilde{W}_u = \widetilde{W} + \sum_{j \text{ incident to } i} w_j (r'_j - r_j)$$

where r_j (resp., r'_j) stores the probability that constraint j is satisfied given that variables $1, \ldots, i-1$ are fixed, and the remaining variables assigned randomly (resp., i is assigned to u and the rest assigned randomly). Letting l be the other variable connected by j, r'_j is computed as follows:

if $l < i$ (i.e. l has been assigned)
then set r'_j to 1 if $(\sigma_l, u) \in R_j$ and 0 otherwise
else set r'_j to the fraction $|\{v \in K : (u, v) \in R_j\}|/k$.

Clearly, the computation of each $\widetilde{W}_u$ takes $O(m_i k)$ time, where m_i is the degree of variable i. Hence, the total time needed is $O(\sum m_i k^2) = O(mk^2)$, which is linear in the size of the input.

Hence, to obtain assignments of large weights, the key design issue is to derive the probability distribution matrix Π such that the *expected weight* is as large as possible.

Consider the most basic probability distribution—the *random* assignment, i.e., for all i and u, we have $p_{i,u} = 1/k$. By linearity of expectation (i.e., expected sum of random variables is equal to the sum of expected values of random variables), the expected weight of the random assignment is given by,

$$\overline{W} = \sum_{j \in M} w_j \cdot s_j = s \sum_{j \in M} w_j.$$

implying that W-CSP can be approximated within absolute ratio s. In the next section, we will present a more clever choice of Π which will give us an improved ratio.

4. Randomized Rounding of Semidefinite Program

In this section, we present our main algorithm. Essentially, the idea is to represent W-CSP as a QIP and apply randomized rounding to its semidefinite programming relaxation. The resulting randomized algorithm is then derandomized into a deterministic algorithm by the method of conditional probabilities.

Given an instance of W-CSP(k), we formulate the corresponding 0/1 QIP instance (Q) as follows. For every variable $i \in V$, define k 0/1 variables $x_{i,1}, \ldots, x_{i,k}$ in (Q) such that i is assigned to u in the W-CSP instance iff $x_{i,u}$ is assigned to 1 in (Q).

$$\begin{aligned} \text{Q}: \text{maximize} \quad & \sum_{j \in M} w_j f_j(x) \\ \text{subject to} \quad & \sum_{u \in K} x_{i,u} = 1 \quad \text{for } i \in V \qquad (1) \\ & x_{i,u} \in \{0,1\} \quad \text{for } i \in V \text{ and } u \in K \end{aligned}$$

In the above formulation, $f_j(x) = \sum_{u,v} c_j(u,v) x_{\alpha_j,u} x_{\beta_j,v}$ encodes the satisfiability of constraint j and hence the objective function gives the weight of the assignment. Inequality (1) ensures that every W-CSP variable gets assigned to exactly one value.

Next, convert this 0/1 QIP into an equivalent QIP (Q′) whose variables takes values $\{-1, +1\}$:

$$\begin{aligned} \text{Q}': \text{maximize} \quad & \sum_{j \in M} w_j f'_j(x) \\ \text{subject to} \quad & \sum_{u \in K} x_0 x_{i,u} = -(k-2) \quad \text{for } i \in V \\ & x_{i,u} \in \{-1,+1\} \quad \text{for } i \in V \text{ and } u \in K \\ & x_0 = +1 \end{aligned}$$

Here, we have $f'_j(x) = \frac{1}{4} \sum_{u,v} c_j(u,v)(1 + x_0 x_{\alpha_j,u})(1 + x_0 x_{\beta_j,v})$. The reason for introducing a normalizing variable x_0 is so that all terms occurring in the formulation are quadratic, which is necessary for the subsequent semidefinite programming relaxation.

Having formulated the W-CSP instance as a QIP, the next step is to find an appropriate relaxation which is polynomial-time solvable. One such candidate is the linear programming relaxation. In [11], it has been shown that linear programming relaxations do not yield a strong bound compared with semidefinite programming relaxations for small domain sizes. In this paper, we will only discuss semidefinite programming relaxations.

The essential idea is to coalesce a quadratic term $x_s x_t$ into a matrix variable $y_{s,t}$. Let Y denote the $(kn+1)\times(kn+1)$ matrix comprising these matrix variables. The resulting relaxation problem (P) is the following:

$$\begin{aligned} \text{P}: \text{ maximize } & \sum_{j\in M} w_j F_j(Y) \\ \text{subject to } & \sum_{u\in K} y_{0,i_u} = -(k-2) \quad \text{for } i \in V \\ & y_{i_u,i_u} = 1 \quad \text{for } i \in V \text{ and } u \in K \\ & y_{0,0} = 1 \\ & Y \text{ symmetric positive semidefinite.} \end{aligned} \tag{2}$$

Here, $F_j(Y) = \frac{1}{4}\sum_{u,v} c_j(u,v)(1+y_{\alpha_{j_u},\beta_{j_v}}+y_{0,\alpha_{j_u}}+y_{0,\beta_{j_v}})$.

(P) is a semidefinite program, which can be solved in polynomial time within an additive factor (see [1]). By a well-known theorem in Linear Algebra, a $t\times t$ matrix Y is symmetric positive semidefinite iff there exists a full row-rank matrix $r\times t$ $(r\le t)$ B such that $Y = B^T B$ (see for example, [8]). One such matrix B can be obtained in $O(t^3)$ time by an incomplete Cholesky's factorization. Since Y has all 1's on its diagonal (by inequality (2)), the decomposed matrix B corresponds precisely to a list of t unit-vectors $X_1,\ldots,X_t$ where any arbitrary column c of B gives the vector X_c. Furthermore, these vectors have the nice property that $X_c\cdot X_{c'} = y_{c,c'}$. The notation $X_i\cdot X_j$ denote the inner product of the vectors X_i and X_j.

Hence, by solving the semidefinite program and performing Cholesky's matrix factorization, we obtain a list of unit vectors, where each vector X_{i_u} corresponds to a relaxation of the variable x_{i_u} in the original problem (Q′). This relaxation becomes evident when we view the allowable values of $x_{i,u}$ (i.e., -1 and $+1$) as unit vectors in one dimension, and the relaxation consists of allowing $x_{i,u}$ to be unit vectors in higher dimension space. The question is therefore how to round these vectors to $\{-1+1\}$. In our approach, this is done in a randomized fashion shown below.

Randomized Algorithm We now propose our randomized algorithm for solving a given W-CSP instance:

1. Solve the corresponding semidefinite program (P) to optimality and obtain an optimal set of vectors X^*.
2. Construct an assignment for the given instance as follows. For each i, assign variable i to value u with probability $1+X_0^*\cdot X_{i,u}^*/2$.

Note that this rounding scheme works for all domain sizes k since the sum of probabilities for each variable i is equal to

$$\frac{1}{2}\sum_{u=1}^{k}(1 + X_0^* \cdot X_{i,u}^*) = 1.$$

An open question is to analyze the worst case performance of this rounding scheme. While we cannot fully settle this question, if we choose a *different* rounding strategy for the cases of $k = 2$ and $k = 3$, we can derive worst-case bounds analytically, as shown below.

Domain Size $k = 2$ For this case, we consider the following rounding strategy:

$$\text{assign variable } i \text{ to value } u \text{ with probability } 1 - \frac{\arccos(X_0^* \cdot X_{i,u}^*)}{\pi}.$$

This rounding strategy has the following intuitive meaning. The normalizing vector X_0^* is associated with the integer value 1. Hence, the smaller the angle between $X_{i,u}^*$ and X_0^* (i.e., the nearer $X_{i,u}^*$ is near to X_0^*), the higher the probability that $x_{i,u}$ would be 1, or equivalently, that i would be assigned to u.

Under this scheme, the expected weight of the probabilistic assignment can be shown to be at least 0.408 times the weight of the optimal solution [11]. The randomized rounding step can be converted into a deterministic algorithm using the technique discussed in Section 3. Hence, we can approximate W-CSP(2) within a worst-case bound of 0.408.

Domain Size $k = 3$ For the above bound to be applied to the case of $k = 3$, the technical difficulty is in ensuring that the sum of probabilities of assigning a variable to the three values is exactly 1. Fortunately, by introducing additional valid inequalities, it is possible to enforce this condition, which we will now explain.

Call two vectors X_1 and X_2 *opposite* if $X_1 = -X_2$. The following lemma is proved in [11]:

Lemma 4.1 ([11]) *Given* 4 *unit vectors* a, b, c, d *if*

$$a \cdot b + a \cdot c + a \cdot d = -1$$
$$b \cdot a + b \cdot c + b \cdot d = -1$$
$$c \cdot a + c \cdot b + c \cdot d = -1$$
$$d \cdot a + d \cdot b + d \cdot c = -1$$

then $a, b, c,$ *and* d *must form two pairs of opposite vectors.* (*QED*)

Consider adding the following set of $4n$ valid equations into (Q′). For all i:

$$x_0(x_{i,1} + x_{i,2} + x_{i,3}) = -1$$
$$x_{i,1}(x_0 + x_{i,2} + x_{i,3}) = -1$$
$$x_{i,2}(x_0 + x_{i,2} + x_{i,3}) = -1$$
$$x_{i,3}(x_0 + x_{i,2} + x_{i,3}) = -1$$

By Lemma 4.1, the relaxation problem (P) will return a set of vectors with the property that for each i, there exists at least one vector $\widetilde{X} \in \{X_{i,1}, X_{i,2}, X_{i,3}\}$ opposite to X_0 while the remaining two are opposite to each other. Noting that $1 - \arccos(X_0 \cdot \widetilde{X})/\pi = 0$, the sum of probabilities of assigning i to the other two values is exactly 1. Thus, we have reduced the case of $k = 3$ to the case of $k = 2$, implying that the approximation ratio of 0.408 also holds for $k = 3$.

5. Computational Experience

In this section, we report our computational experience. Our experiments are conducted on the SUN Sparc 10 UNIX workstation. Random numbers are generated using the standard UNIX `long random()` function, initialized with a random seed which depends on the time of the day.

We naturally wanted to test our algorithms on hard W-CSP instances. However, we learnt that for PCSP, there is no localized region of hard problems—hard problems are located throughout the instance space, with difficulty increasing with increasing edge density of the constraint graph, increasing tightness of constraints, and naturally, increasing domain size [17]. Our main concern is the performance of our algorithm against other incomplete algorithms which run in polynomial time. The measure of performance is the approximation ratio. Since it is time-consuming to compute optimal solutions for reasonably large instances, we first experiment on *satisfiable* instances whose optimal value is always the sum of all edge weights. This allows us to compute the approximation ratio without obtaining optimal solutions.

Generation of Random Instances With the above considerations, we generate W-CSP instances of n variables from a distribution parameterized by the *edge probability* $0 \leq q \leq 1$ and the *consistency probability* $0 \leq \lambda \leq 1$ according to the following rules:

1. Generate a random graph G of n nodes such that an edge (i.e., constraint) exists between any two variables with probability q.
2. To generate a satisfiable instance, we first generate a random assignment $\hat{\sigma}$. For each edge (i_1, i_2) in G, construct the constraint relation as follows. Insert the value pair $(\hat{\sigma}_{i_1}, \hat{\sigma}_{i_2})$ with probability 1 and all other $k^2 - 1$ pairs with probability λ. To generate a nonsatisfiable instance, we simply insert the value pair $(\hat{\sigma}_{i_1}, \hat{\sigma}_{i_2})$ with probability λ.
3. Edge weights are randomly generated in the range [0..999].

This generation method has been used by others and an online (unweighted) implementation is in [15]

Algorithms Four algorithms are compared.

1. Greedy LS. This represents a naive algorithm for benchmarking purposes. In this algorithm, we first generate an initial assignment greedily, i.e., arrange the variables

in a linear order and assign each in sequence the value that maximizes the weighted sum of satisfied constraints. Then, perform a *basic* min-conflict local search, i.e., move from one assignment to another based on the **Min-Conflict Heuristic** until a local optimal is reached (i.e., the Heuristic fails). The **Min-Conflict Heuristic** is defined as follows. Select a variable-value pair that minimizes the total weight of conflicts, i.e., maximizes the weight of satisfied constraint (break ties randomly). Fail when no such variable-value pair can be found that decreases the weight of conflicts.

2. Random LS. This is a randomized and iterative version of the min-conflict local search scheme. First, generate an initial assignment randomly. Then, perform a prescribed number of iterations of the following procedure. Randomly select a variable in conflict and re-assign a value which minimizes the total weight of conflicts (break ties randomly). Note that the chosen variable is re-assigned its current value if and only if no other values can decrease the total weight of conflicts. We set the number of iterations to be 4000, so as to match the CPU time required by our randomized rounding scheme.

3. RR LS. This is our rounding algorithm described in Section 4, followed by a *basic* min-conflict local search (see above). To solve the semidefinite program, we use the SDPA solver written by Fujisawa and Kojima [5].

Experiment 1 In the first set of experiments, we fix $n = 64$ and $k = 2$ and generate random satisfiable instances by the abovementioned method. We consider three edge densities (sparse, medium, and dense; $m = \binom{n}{2}$ refers to the total number of edges) and for each density, we vary the consistency probability from 0.1 to 0.9. For each case, 10 random satisfiable instances are generated and solved respectively by the three algorithms. The respective mean approximation ratios are obtained. Table 1 gives the outcome of the experiment. Figures are rounded to three decimal places and $1.000(-)$ is used to denote a value which is rounded to 1.000.

Experiment 2 In the second set of experiments, we fix $n = 20$ and $k = 5$. The same scenario as Experiment 1 is repeated. Table 2 gives the outcome of the experiment.

Observations (Satisfiable Instances)

1. Greedy LS performs well on dense instances, but not so well on sparse ones.

2. Random LS performs consistently well on all instances, achieving at least 98% optimality for $k = 2$ and 91% optimality for $k = 5$.

3. RR LS outperforms other approaches in all cases, achieving 99% optimality for $k = 2$ and 96% optimality for $k = 5$. Furthermore, we observe that, in all cases, the standard deviation corresponding to each mean approximation ratio is smaller than those of Greedy LS and Random LS. This means that RR LS gives *consistently* good approximation solutions for the instances tested.

Table 1. Experiment 1

Edge prob (q)	Consistency prob (λ)	Mean approximation ratios		
		Greedy LS	Random LS	RR LS
Sparse ($2n/m$)	0.10	0.935	0.999	1.000
	0.30	0.916	0.994	0.995
	0.50	0.930	0.987	0.994
	0.70	0.974	0.989	0.992
	0.90	0.996	0.998	0.999
Medium ($n \log n/m$)	0.10	1.000	1.000	1.000
	0.30	1.000	1.000	1.000
	0.50	0.999	1.000	1.000
	0.70	0.988	0.997	1.000(−)
	0.90	0.991	0.996	0.999
Dense ($n^2/3m$)	0.10	1.000	1.000	1.000
	0.30	1.000	1.000	1.000
	0.50	1.000	1.000	1.000
	0.70	1.000	1.000	1.000
	0.90	0.997	0.999	1.000(−)

4. The average CPU time required by our implementation of RR LS is 22.2 seconds for each instance in Experiment 1 and 10.5 seconds for each instance in Experiment 2. The CPU time required by our implementation of Random LS is dependent on the number of iterations. For 4000 iterations, Random LS consumes equal amount of CPU time as RR LS. The CPU time for Greedy LS is negligible (less than 1 second).

Table 2. Experiment 2

Edge prob (q)	Consistency prob (λ)	Mean approximation ratios		
		Greedy LS	Random LS	RR LS
Sparse ($2n/m$)	0.10	0.752	0.968	1.000
	0.30	0.753	0.914	0.960
	0.50	0.911	0.957	0.975
	0.70	0.981	0.996	0.999
	0.90	1.000	1.000	1.000
Medium ($n \log n/m$)	0.10	0.823	1.000	1.000
	0.30	0.751	1.000	1.000
	0.50	0.889	0.927	0.968
	0.70	0.961	0.980	0.982
	0.90	0.999	0.999	1.000
Dense ($n^2/3m$)	0.10	0.893	1.000	1.000
	0.30	0.919	1.000	1.000
	0.50	0.949	0.929	0.984
	0.70	0.927	0.963	0.966
	0.90	0.998	0.999	1.000(−)

From these observations, we make the following statements of comparison:

1. With the same local search (i.e., basic min-conflict) as the post-processing phase, RR LS far outperforms Greedy LS in all cases. This seems to suggest that a good initial solution obtained by randomized rounding is able to lead to a better solution than the greedy initial solution. However, on closer observation, we saw that RR LS's initial solution already outperforms the final solution of Greedy LS.
2. With the same amount of CPU time spent on both Random LS and RR LS, the latter gives better solutions. This suggests that by investing in some extra algorithmic rigour, we can derive solutions which a simplistic local search cannot achieve, even when given a large number of iterations.

Experiments 3 and 4 In the third and fourth sets of experiments, we generate non-satisfiable instances and measure the *absolute* ratios (i.e., the denominator is the total number of constraints instead of the optimal number of satisfiable constraints). Absolute ratios are measured because it is intractible to obtain optimal solutions via exhaustive search with large n. Again, we fix $n = 64$, $k = 2$ and $n = 20$, $k = 5$ respectively. The same scenario as Experiment 1 is repeated. Tables 3 and 4 give the outcome of the experiments.

Observations (Non-Satisfiable Instances) The superior performance of RR LS portrayed in Experiments 1 and 2 do not seem to carry over to Experiments 3 and 4. In general, however, RR LS still performs better than the other two approaches. Particularly, Random LS performs slightly better than RR LS in only 6 out of the 30 ratios.

Table 3. Experiment 3

Edge prob (q)	Consistency prob (λ)	Mean absolute ratios		
		Greedy LS	Random LS	RR LS
Sparse ($2n/m$)	0.10	0.244	0.277	0.264
	0.30	0.555	0.571	0.579
	0.50	0.769	0.774	0.792
	0.70	0.904	0.936	0.933
	0.90	0.984	0.992	0.997
Medium ($n \log n/m$)	0.10	0.214	0.215	0.218
	0.30	0.472	0.479	0.479
	0.50	0.662	0.669	0.661
	0.70	0.846	0.852	0.856
	0.90	0.977	0.980	0.981
Dense ($n^2/3m$)	0.10	0.162	0.161	0.163
	0.30	0.387	0.389	0.390
	0.50	0.592	0.593	0.594
	0.70	0.784	0.785	0.786
	0.90	0.950	0.957	0.952

Table 4. Experiment 4

Edge density (q)	Consistency prob (λ)	Mean absolute ratios		
		Greedy LS	Random LS	RR LS
Sparse ($2n/m$)	0.10	0.455	0.490	0.503
	0.30	0.752	0.799	0.825
	0.50	0.909	0.921	0.963
	0.70	0.977	0.997	0.996
	0.90	0.984	1.000	1.000
Medium ($n \log n/m$)	0.10	0.367	0.390	0.395
	0.30	0.659	0.652	0.662
	0.50	0.822	0.848	0.870
	0.70	0.956	0.964	0.977
	0.90	0.997	0.994	1.000
Dense ($n^2/3m$)	0.10	0.298	0.326	0.313
	0.30	0.597	0.605	0.612
	0.50	0.787	0.794	0.809
	0.70	0.933	0.941	0.943
	0.90	0.996	0.999	1.000

Acknowledgments

I would like to thank Osamu Watanabe for discussions on theoretical results, Nobuhiro Yugami for discussions on computational results, Richard Wallace for comments on early drafts of this paper, and the two anonymous referees for their insightful comments.

Note

1. For standard unweighted CSP, this measure is often known as *looseness*.

References

1. Alizadeh, F. (1995). Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM Journal of Optimization*, 5(1): 13–51.
2. Alon, N., & Spencer, J. (1992). *The Probabilistic Method*. Wiley Interscience Ser. Disc. Math. and Optimiz., Wiley, New York.
3. Freuder, E. C. (1989). Partial constraint satisfaction. In *Proceedings of the International Joint Conference Artificial Intelligence (IJCAI)*, pages 278–283, Detroit, MI.
4. Freuder, E. C., & Wallace, R. J. (1992). Partial constraint satisfaction. *Artificial Intelligence*, 58(1–3): 21–70.
5. Fujisawa, K., & Kojima, M. (1995). Sdpa (semidefinite programming algorithm) user's manual. Technical Report B-308, Dept. Information Science, Tokyo Inst. of Technology. Online implementation available at ftp.is.titech.ac.jp under directory /pub/OpsRes/software/SDPA.

6. Galinier, P., & Hao, J.-K. (1997). Tabu search for maximal constraint satisfaction problems. In *Proceedings of the Third International Conference Principles and Practice of Constraint Programming (CP)*, pages 196–208. Springer Verlag Lecture Notes Comp. Sci. (1330).
7. Goemans, M. X., & Williamson, D. P. (1994). Approximation algorithms for MAX CUT and MAX 2SAT. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 422–431. Full version in *J. ACM*, 42: 1115–1145.
8. Lancaster, P., & Tismenetsky, M. (1985). The *Theory of Matrices*. Academic Press, Orlando, FL.
9. Larrosa, J., & Meseguer, P. (1995). Optimization-based heuristics for maximal constraint satisfaction. In *Proceedings of the First International Conference Principles and Practice of Constraint Programming (CP)*, pages 103–120. Springer Verlag, Lecture Notes Comp. Sci.
10. Lau, H. C. (1995). Approximation of constraint satisfaction via local search. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures (WADS)*, pages 461–472. Springer Verlag Lecture Notes Comp. Sci. (955).
11. Lau, H. C., & Watanabe, O. (1996). Randomized approximation of the constraint satisfaction probem. *Nordic Journal of Computing*, 3: 405–424.
12. Mahajan, S., & Ramesh, H. (1995). Derandomizing semidefinite programming based approximation algorithms. In *Proceedings of the 36th IEEE Symposium on Found. of Comp. Sci.*, pages 162–168.
13. Raghavan, P., & Thompson, C. D. (1987). Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4): 365–374.
14. Tsang, E. (1993). *Foundations of Constraint Satisfaction*. Academic Press.
15. van Beek, P. On-line C-programs available at http://ai.waterloo.ca/~vanbeek/software/software.html.
16. Voudouris, C., & Tsang, E. (1995). Partial constraint satisfaction problems and guided local search. Technical Report No. CSM-250, Dept. Computer Science, University of Essex.
17. Wallace, R. J. (1995). Personal communication.
18. Wallace, R. J. (1995). Directed arc consistency preprocessing as a strategy for maximal constraint satisfaction. In M. Meyer, ed., *Constraint Processing*, pages 121–138. Springer Verlag Lecture Notes Comp. Sci. (923).
19. Wallace, R. J. (1996). Analysis of heuristic methods for partial constraint satisfaction problems. In *Proceedings of the Second International Conference Principles and Practice of Constraint Programming (CP)*, pages 482–496. Springer Verlag Lecture Notes Comp. Sci. (1118).
20. Wallace, R. J., & Freuder, E. C. (1993). Conjunctive width heuristics for maximal constraint satisfaction. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 762–768.
21. Wallace, R. J., & Freuder, E. C. (1995). Heuristic methods for over-constrained constraint satisfaction problems. In *CP95 Workshop on Over-Constrained Systems*.