Tabu Search for Maximal Constraint Satisfaction Problems

Philippe Galinier and Jin-Kao Hao

LGI2P
EMA-EERIE
Parc Scientifique Georges Besse
F-30000 Nîmes
France
email: {galinier, hao}@eerie.fr

Abstract. This paper presents a Tabu Search (TS) algorithm for solving maximal constraint satisfaction problems. The algorithm was tested on a wide range of random instances (up to 500 variables and 30 values). Comparisons were carried out with a min-conflicts+random-walk (MCRW) algorithm. Empirical evidence shows that the TS algorithm finds results which are better than that of the MCRW algorithm.the TS algorithm is 3 to 5 times faster than the MCRW algorithm to find solutions of the same quality.

Keywords: Tabu search, constraint solving, combinatorial optimization.

1 Introduction

A finite Constraint Network (CN) is composed of a finite set X of variables, a set D of finite domains and a set C of constraints over subsets of X. A constraint is a subset of the Cartesian product of the domains of the variables involved that specifies which combinations of values are compatible. A CN is said to be binary if all the constraints have 2 variables. Given a CN, the Constraint Satisfaction Problem (CSP) consists in finding one or more complete assignments of values to the variables that satisfy all the constraints [13] while the Maximal Constraint Satisfaction Problem (MCSP) is an optimization problem consisting in looking for an assignment that satisfies the maximal number of constraints [4]. Known to be NP-complete (NP-hard) in general, both MCSP and CSP are of great importance in practice. In fact, many applications related to allocation, assignment, scheduling and so on can be modeled as a CSP or a MCSP.

Methods for solving CSP include many *complete* algorithms based on backtracking and filtering techniques [19] and *incomplete* ones based on repair heuristics such as Min-conflicts [14]. Similarly, methods for solving MCSP include *exact* algorithms based on branch-and-bound techniques [4, 12, 20] and *approximation* ones based on the above mentioned repair heuristics [21].

The main advantage of an *exact* (*complete* for CSP) method is its guarantee of optimality (completeness for CSP). The main drawback of such a method remains the time necessary to compute large scale instances. On the contrary, repair techniques constitute an interesting alternative to deal with instances of very large size although neither optimality nor completeness is guaranteed.

Repair methods belong in fact to a more general class of methods called Local Search (LS). Local search, which is based on the notion of neighborhood, constitutes a powerful approach for tackling hard optimization problems [16, 10]. Starting with an initial configuration, a typical local search method replaces iteratively the current configuration or solution by one of its neighbors until some stop criteria are satisfied; for example, a fixed number of iterations is reached or a sufficiently good solution is found. Well-known examples of LS methods include various hill-climbers, simulated annealing (SA) [11] and Tabu search (TS) [5].

TS is generally considered to be one of the most promising methods in combinatorial optimization and already shows its power for solving many hard problems including the maximal satisfiability [6] and graph-coloring problem [8, 3]. In this study, we are interested in applying TS to solve MCSP and try to answer the following question: is TS a competitive method for this problem ?

This paper presents a TS algorithm for solving MCSP. In order to evaluate the effectiveness of the TS algorithm, extensive experiments are carried out on a wide range of random instances (up to 500 variables and 30 values). Experimental results were compared with a min-conflicts algorithm combined with random walk (MCRW), which is considered to one of the most successful method for MCSP [21].

The paper is organized as follows: after a brief review of repair methods (Section 2), we present Tabu Search and its adaptation to MCSP (Section 3). Then we define the context and method of the experimentation, followed by comparative results between TS and MCRW (Section 4). We conclude the paper with some conclusions and indications about our ongoing work (Section 5).

2 Repair Methods for MCSP

An instance of an optimization problem (S, f) is defined by a set S (search space) of configurations and a cost function f: $S \rightarrow R$ (R being the set of real numbers). Solving such an instance consists in finding a configuration $s \in S$ that has the minimal (or maximal) value of the cost function f.

Given a CN $< X, D, C >$ representing respectively the set of distinct variables, value domains, and constraints, MCSP is the optimization (minimization) problem defined by:

- The set of configurations S is the set of all the complete assignments s defined by $s = \{< V_i, v_i > \mid V_i \in X \text{ and } v_i \in D_i\}$. Clearly the cardinality of the search space S is equal to the product of the sizes of the domains, i.e. $\prod_{i=1}^{n} |D_i|$.
- The cost $f(s)$ of a configuration s is the number of constraints violated by s.

A typical repair method for MCSP begins with an inconsistent complete assignment, often generated randomly, and then repairs iteratively the current solution. To carry out a repair, a two step process is usually used: first choose a *variable* and then choose a new value for the chosen variable. Many heuristics are possible for both choices leading thus to different repair methods [7]. One example for choosing a value for a given variable is the Min-Conflicts (MC) heuristic [14].

– **Min-Conflicts Heuristic**: For a given conflicting variable[1], pick a value which minimizes the number of violated constraints (break ties randomly); if no such value exists, pick randomly one value that does not increase the number of violated constraints (the current value of the variable is picked only if all the other values increase the number of violated constraints).

A MC-based repair algorithm may consist simply in iterating the MC heuristic. The solving power of such a MC algorithm is limited because it cannot go beyond a local optimum. However, its performance can be largely improved when some noise strategies are introduced in MC. This is exemplified by MCRW which is a MC algorithm combined with the *random-walk* strategy [17]: for a given conflicting variable, pick randomly a value with probability p, apply the MC heuristic with probability $1-p$. More precisely, a MCRW algorithm can be defined as in Figure 1.

```
MCRW algorithm

begin
    generate a random configuration $s$
    $nb\_iter := 0$
    $nb\_moves := 0$
    while $f(s) > 0$ and $nb\_moves < max\_moves$ do
        if probability $p$ verified then
            choose randomly a variable $V$ in conflict
            choose randomly a value $v'$ for $V$
        else
            choose randomly a variable $V$ in conflict
            choose a value $v'$ that minimises the number of conflicts for $V$
            (the current value is chosen only if all the other values increase
            the number of violated constraints)
        if $v'$ is different from the current value of $V$ then
            assign $v'$ to $V$
            $nb\_moves := nb\_moves + 1$
        $nb\_iter := nb\_iter + 1$
    output(s);
end
```

Figure 1: The MCRW algorithm

[1] A variable is said conflicting if it is involved in some unsatisfied constraints.

This algorithm is controlled by the random probability p, it should be clear that the value for this parameter has a big influence on the performance of the algorithm. The tuning of this parameter is explained in Section 4.3.

An iteration leading to a new configuration different from the current one is called a *move*. Note that for MCRW, many iterations do not lead to a move (see Section 4.1). An optimization is introduced as follows. If an iteration fails to lead to a move for a conflicting variable V, V will not be uselessly considered for the next iterations until a move is effectively carried out.

There are other strategies to help MC to escape from local optima, we can mention for instance Breakout [15] and EFLOP [22].

Recently, some empirical studies have been reported which compare the above mentioned repair methods for solving constraint problems [7, 21]. Experimental evidence shows that MCRW is among the most powerful methods of the repair family. Consequently, MCRW is used in this paper as our reference to evaluate the performance of the TS algorithm.

3 Tabu Search for MCSP

3.1 Tabu Search

This section gives a very brief review of Tabu Search (TS), emphasizing the most important features which have been implemented in our TS algorithm for MCSP. Instructive presentations of TS are given in [5], including many pointers to other applications.

Like any LS method, TS needs three basic components: a *configuration structure*, a *neighborhood function* defined on the configuration structure, and a *neighborhood examination mechanism*. The first component defines the search space S of the application, the second one associates with each point of the search space a subset of S while the third one prescribes the way of going from one configuration to another. A typical TS procedure begins with an initial configuration in the search space S and then proceeds iteratively to visit a series of locally best configurations following the neighborhood. At each iteration, a *best* neighbor *s'* $\in$ *N(s)* is sought to replace the current configuration *s* even if *s'* does not improve the current configuration in terms of the value of the cost function.

This iterative process may suffer from cycling and get trapped in local optima. To avoid the problem, TS introduces the notion of *Tabu lists*, one of the most important components of the method.

A tabu list is a special short term memory that maintains a selective history *H*, composed of previously encountered solutions or more generally pertinent attributes of such solutions. A simple TS strategy based on this short term memory *H* consists in preventing solutions of *H* from being reconsidered for fonext k iterations (k, called tabu tenure, is problem dependent). Now, at each iteration, TS searches for a best neighbor from this dynamically modified neighborhood *N(H,s)*, instead of from *N(s)* itself. Such a strategy prevents Tabu from being trapped in short term cycling and allows the search process to go beyond local optima.

Tabu restrictions may be overridden under certain conditions, called *aspiration criteria*. Aspiration criteria define rules that govern whether a solution may be included in *N(H,s)* if the solution is classified tabu. One widely used aspiration criterion consists of removing a tabu classification from a move when the move leads to a solution better than that obtained so far. Aspiration constitutes an important element of flexibility in TS.

TS uses an aggressive search strategy to exploit its neighborhood. Therefore, it is crucial to have special data structures which allow a fast updating of move evaluations, and reduce the effort of finding best moves. Without this kind of technique, the efficiency of TS may be compromised.

There are other interesting and important techniques available such as intensification and diversification. In this paper, we show that a TS algorithm based on the above mentioned elements may be very efficient and robust for MCSP.

3.2 A TS Algorithm for MCSP

Definition of the neighborhood

We choose for the TS algorithm the following neighborhood function $N : S \rightarrow 2^S$: for each configuration s in S, $s' \in N(s)$ if and only if s and s' are different at the value of a *single conflicting* variable. In other words, a neighbor of a configuration s can be obtained by changing the current value of a conflicting variable in s. Note that the size of this neighborhood $|N(s)|$ varies during the search according to the number of conflicting variables in s.

Tabu list and definition of attributes

A move for MCSP corresponds to changing the value v of a conflicting variable V to another value v', and therefore can be characterized by a couple (attribute) $< variable, value >$. Consequently, when the solution s moves to $s' \in N(s)$ by replacing the current value v of V with a new one v', the $< V, v >$ is classified tabu for the next k iterations. In other words, the value v is not allowed to be re-assigned to V during this period. Like the random probability p for the MCRW algorithm, the tabu tenure k has a big influence on the performance of the TS algorithm. The tuning of this parameter is explained in Section 4.3.

In order to implement the tabu list, we use a $|X|*|D|$ matrix T. Each element of T corresponds to a possible move for the current configuration. When a move is done, the corresponding element of the matrix is set to the current number of iterations plus the tabu tenure k. In this way, it is very easy to know if a move is tabu or not by simply comparing the current number of iterations with that memorized in T.

Aspiration function

The aspiration criterion consists of removing a tabu classification from a move when the move leads to a solution better than the best obtained so far.

Solution evaluation and neighborhood examination

For MCSP, it is possible to develop special data structures to find quickly a best neighbor among the given neighborhood. The main idea, inspired by a technique proposed in [3], is based on a two dimensional table $|X|*|D|$ γ: if v is the current value of V, then $\gamma[V,v]$ indicates the the current number of conflicts for V; for each v' different from v, $\gamma[V,v']$ indicates the the number of conflicts for V if v' is assigned to V. Each time a move is executed, only the affected elements of the table are updated accordingly. In this way, the cost for each move is constantly available and a best move can be found quickly.

This technique is also used by our MCRW algorithm to find quickly a best move among the values of a given (conflicting) variable. In particular, with this data structure, iterations which do not lead to a move will have a negligible cost.

We give in Figure 2 the TS algorithm integrating the above elements.

```
Tabu algorithm
begin
    generate a random configuration $s$
    $nb\_iter := 0$
    initialize randomly the tabu list
    while $f(s) > 0$ and $nb\_iter < max\_iter$ do
        choose a move $< V, v' >$ with the best performance among the non-tabu moves
        and the moves satisfying the aspiration criteria
        introduce $< V, v >$ in the tabu list, where $v$ is the current value of $V$
        remove the oldest move from the tabu list
        assign $v'$ to $V$
        $nb\_iter := nb\_iter + 1$
    output(s);
end
```

Figure 2: The Tabu Search algorithm

Each iteration of TS consists in examining all the values of all the conflicting variables in the current solution s and then carrying out a best move. Unlike MCRW, each iteration modifies a variable and the number of moves carried out by the algorithm is exactly the same as the number of iterations.

4 Experimentation and Results

In this section, we present comparative results between TS and MCRW over several classes of MCSP instances.

4.1 Comparison Criteria

In this study, we choose two criteria to compare TS and MCRW. The first one is the quality of solution measured by the best cost value, *i.e.* the *number of violated constraints,* that a method can find. The second criterion is the computing effort needed by an algorithm to find its best solutions or solutions of a given quality. This second criterion is measured by the *number of moves* and the *running time.*

Note that the number of moves instead of iterations is used in the second criterion. The reason for this choice is the following. For TS, each move coincides with an iteration. But for MCRW, the number of moves is in general much lower than the number of iterations, because an iteration of MCRW does not lead to a move if the current value of the chosen variable is picked. In our implementation, the data structure shared by TS and MCRW has the property that an iteration which does not lead to a move is much less costly than an iteration leading to a move. Therefore, counting iterations instead of moves, will put MCRW at a disadvantage, especially if the iterations which do not lead to a move are frequent. As we explain later, this is indeed the case for MCRW.

4.2 MCSP Instances

Let us note that a *satisfaction* problem (such as CSP and SAT) is very different from an *optimization* problem (such as MCSP and MAX-SAT). There exist some interesting results such as the phase transition phenomenon which seperates under-constrained problems and over-constrained ones [1, 9, 18, 2]. Under-constrained problems tend to have many solutions. Therefore, it is usually easy to find a satisfiable assignment which is also an optimal solution for the optimization problem. Over-constrained problems tend to be easy to be proven unsatisfiable. This means only that they are easy from the point of view of satisfiability, but nothing is known about the difficulty of finding a minimal cost solution from the point of view of optimization.

In summary, for the satisfaction problem, hard instances tend to be around the phase transition. However, for the optimization problem, it is not possible to determine whether a given instance is difficult to solve or not except for under-constrained instances.

In our expriments, we choose a classical and simple binary MCSP model and evaluate the algorithms with instances coming from sufficiently diversified classes.

More precisely, the MCSP model used depends on 4 parameters: the number n of variables, the size d of domains (all domains have the same size), the density *p1* and the tightness *p2* of constraints. A quadruple (n, d, p_1, p_2) defines a particular class. An instance of a given class is built by choosing randomly $p_1.n.(n-1)/2$ constraints among the $n.(n-1)/2$ different pairs of variables and, for each constraint, $p_2.d^2$ forbidden tuples among the d^2 possible tuples. Different instances of a same class can be generated using different random seeds.

Classes used in this work are taken from the following sizes: small (n/d = 50/10), medium (n/d = 100/15), large (n/d = 250/25) and very large (n/d =

300/30, 500/30). For each size, we choose one or several values for the couple (*p1*,*p2*) in such a way that the instances of the class are not easily satisfiable and have a cost value smaller than 30.

4.3 Parameter tuning for TS and MCRW

The performance of TS and MCRW is greatly influenced by some parameters: the size of tabu list *tl* and the random walk probability *p*. We use the following procedure to determine the appropriate values for these parameters for each class of instances. First, a preliminary study determined the following ranges of parameter values: $10 \leq tl \leq 35$ and $0.02 \leq p \leq 0.1$. Then, different discrete values between these ranges were further tested as follows. For TS (respectively MCRW) each of the values {10, 15, 20, 25, 30, 35, 40, 45} ({0.02, 0.03, 0.04, 0.05, 0.07, 0.1, 0.2, 0.3, 0.4} for MCRW) was used to run 50 times on the class (10 instances, 5 runs per instance, each run being limited to 50,000 moves) and the best value identified for the class.

4.4 Results

Problem	Tabu				MCRW					Tabu-MCRW	
	tl	cost value *f*			*p*	%	cost value *f*				
		min	avg	max			min	avg	max	min	avg
50.10.10.60.0	15	4	4	4	0.05	42	4	4	4	0	0
50.10.10.70.0	15	14	14	14	0.05	35	14	14	14	0	0
50.10.30.30.0	15	5	5.08	6	0.05	38	5	5.5	6	0	-0.42
50.10.30.35.0	15	15	15	15	0.05	33	15	15.18	16	0	-0.18
50.10.50.20.0	15	8	8	8	0.05	36	8	8.26	9	0	-0.26
100.15.10.40.0	15	0	1.2	2	0.03	35	1	2.33	4	-1	-1.13
100.15.10.45.0	25	8	9.72	12	0.03	30	8	11.02	13	0	-1.3
100.15.10.50.0	30	20	21.62	24	0.03	26	20	23.18	27	0	-1.55
100.15.30.15.0	10	0	0	0	0.03	27	0	0.06	1	0	-0.06
100.15.30.20.0	20	19	20.78	23	0.03	25	19	22.3	25	0	-1.51
250.25.03.55.0	40	6	8.42	12	0.02	28	6	12.36	15	0	-3.93
250.25.07.30.0	30	7	11.68	14	0.03	34	10	14.66	19	-3	-2.97
250.25.10.21.0	25	3	4.88	7	0.03	33	3	6.9	10	0	-2.02
250.25.10.22.0	30	8	10.86	14	0.03	34	10	14.17	19	-2	-3.31
250.25.10.23.0	35	15	19.18	22	0.02	32	19	22.96	27	-4	-3.77
300.30.03.50.0	45	5	10.1	15	0.03	28	8	14.9	22	-3	-4.79
300.30.05.35.0	35	9	13.96	17	0.02	28	14	18	23	-5	-4.04
300.30.07.25.0	25	1	3.48	6	0.02	32	4	7.04	11	-3	-3.56
500.30.04.25.0	30	0	1.56	3	0.02	36	3	5.97	13	-3	-4.41

Table 1. Comparative results of TS and MCRW, maximum moves = 100,000

Table 1 gives a summary of the results of TS and MCRW for the instances we have tested in terms of quality of the solutions. To obtain these results, both algorithms were run 50 times on each instance, each run being given a maximum of 100,000 moves. The parameter of each algorithm (the size of tabu list *tl* and the random-walk probability *p*) is fixed according to the best value found during the parametric study. For each algorithm, we give the minimum, average and maximum value of the cost function. For MCRW, we also give the average percentage of iterations that lead effectively to moves (column %). The last two columns indicate the difference between the cost function of Tabu and MCRW, in minimum and average.

From the data of Table 1, the following observations may be made. For small instances, both methods obtain the same minimum for the cost function. For medium and large instances, TS obtains better results for 4 out of 10 instances (a smaller cost value, -1 to -4). For the 4 very large instances, TS obtains much better results (-3 to -5). If we look at the average cost of the two algorithms, we see that TS obtains better results than MCRW for 17 out of 19 instances and the same results for the 2 others (seemingly easy ones). Finally, note that less than half (25-42%) of MCRW iterations lead to moves (column %).

Table 2 gives more details about the performance of TS and MCRW. This table shows different values of the cost function together with the number of successful runs, the minimal, average and maximal number of moves (for the successful runs). The last column gives the ratio between the average number of moves between MCRW and TS, and hence indicates how much TS is faster than MCRW (in terms of number of moves). For instance, the first line means that both algorithms reach the cost *f=15* at each of 50 runs, but TS needs on average a number of moves 3.49 times smaller than MCRW. Note that although only one instance is showed here, other instances have very similar behaviors.

Problem	f	Tabu				MCRW				MCRW/Tabu
		succ.	#moves			succ.	#moves			#moves
			min	avg	max		min	avg	max	avg
300.30.07.25.0	15	50	1656	4299.7	7850	50	6013	15022.2	33009	3.49
	14	50	1918	4864.6	9731	50	6324	17051.6	39896	3.5
	13	50	2010	5622.5	11330	50	6367	21506.4	40769	3.82
	12	50	2018	6755.7	15415	50	8694	25244.7	68226	3.73
	11	50	3326	8545	24332	50	8701	33708.5	75467	3.94
	10	50	4513	10039.8	25924	49	9399	41296.3	96774	-
	9	50	4738	12156.1	26545	45	9438	46655.4	99023	-
	8	50	6293	15773.7	39941	37	17335	53922.2	98593	-
	7	50	6306	22562.5	59051	34	20969	63521.8	95867	-
	6	50	10722	31382.2	70505	20	29033	69506.3	99111	-
	5	49	12764	43022.3	93673	9	51662	70315.4	88173	-
	4	40	18585	52437.2	99010	4	76503	85380	96824	-
	3	26	20799	64812.5	96686	0	-	-	-	-
	2	10	57146	79492.5	97850	0	-	-	-	-
	1	1	94525	94525	94525	0	-	-	-	-
	0	0	-	-	-	0	-	-	-	-

Table 2. More detailed comparative results of TS and MCRW, maximum moves = 100,000 for TS and MCRW

In order to see if MCRW can catch up with TS if MCRW is given more moves, a second experiment was performed with MCRW on 2 medium and 2 large instances. In this experiment, the number of moves is extended from 100,000 to 500,000. Table 3 shows the results of this experiment for 300.30.07.25.0 as in Table 2.

From Table 2, we may make the following remarks about this instance. The minimum cost value that an algorithm can reach at each run is 6 for TS and 11 for MCRW. The minimum cost value reached (at least once) is 1 for TS and 4 for MCRW. For the values that both methods could reach at each run ($f \geq 11$), TS is on average about 3 to 4 times faster than MCRW (in terms of number of moves). For the values between 10 and 4, TS has always a higher number of successful runs than MCRW.

Recall that for MCRW, only iterations leading to a real move are counted. Since such iterations leading to a move represent only 25-50% according to the instance (see Table 1), MCRW requires a number of *iterations* 10 to 15 times higher than that of TS.

Data similar to those of Table 2 are available for all the other instances. From the data, we observe very similar behavior. TS is on average 3 to 5 times faster (in terms of number of moves) than MCRW to reach a given cost value and this factor remains stable across all instances and for different cost values of a given instance.

Problem	f	Tabu				MCRW				MCRW/Tabu
		succ.	#moves			succ.	#moves			#moves
			min	avg	max		min	avg	max	avg
300.30.07.25.0	11	50	3326	8545	24332	50	11739	28242	77551	3.3
	10	50	4513	10039.8	25924	50	11789	39613.3	104638	3.94
	9	50	4738	12156.1	26545	50	15019	48761	107704	4.01
	8	50	6293	15773.7	39941	50	15027	63067.7	132980	3.99
	7	50	6306	22562.5	59051	50	15308	86265.8	227572	3.82
	6	50	10722	31382.2	70505	50	16239	133856	355128	4.26
	5	49	12764	43022.3	93673	47	33298	178486	415849	-
	4	40	18585	52437.2	99010	35	85267	236837	461983	-
	3	26	20799	64812.5	96686	25	106969	304038	493080	-
	2	10	57146	79492.5	97850	12	132130	339752	484847	-
	1	1	94525	94525	94525	1	425855	425855	425855	-
	0	0	-	-	-	0	-	-	-	-

Table 3. More detailed comparative results of TS and MCRW, maximum moves = 100,000 for TS, 500,000 for MCRW

From Table 3, we observe that MCRW (with a maximum of 500,000 moves) obtains solutions comparable to those of TS (with a maximum of 100,000 iterations) in terms of quality. In terms of moves required by MCRW and TS, the ratio remains the same as before, *i.e.* around 3.5 to 4.

Other experiments have been carried out where the two algorithms were run 5 times each with a much larger number of moves (2,000,000) on these instances. Similar results have been observed.

Recall that there are two big differences between the TS and MCRW algorithms. The first one is that TS uses a tabu list while MCRW performs random walks. The second one is that the two algorithms examine the neighborhood according to two different strategies: TS looks at all the conflicting variables while MCRW looks only at a single variable. In other words, MCRW examines much fewer neighbor solutions than TS does at each iteration. Therefore, one may ask if this second factor is responsible for the difference of performance between these two algorithms. In order to see if this is the case, we tested a third algorithm which examines at each iteration all the conflicting variables like in TS.

This new algorithm, that we called Steepest Descent Random Walk (SDRW), proceeds as follows. At each iteration, it performs a random walk with probability p, and a "Steepest Descent" with probability $1-p$, *i.e.*, it seeks a best possible move among those which do not increase the cost function.

The SDRW algorithm was tested on the five instances of size $n = 100$ in the same conditions as for TS and MCRW. Results are presented in Table 4.

Problem	Tabu				SDRW					Tabu-SDRW	
	tl	cost value *f*			*p*	%	cost value *f*				
		min	avg	max			min	avg	max	min	avg
100.15.10.40.0	15	0	1.2	2	0.45	90	1	4.1	7	1	-2.9
100.15.10.45.0	25	8	9.72	12	0.45	90	10	14.24	18	2	-4.52
100.15.10.50.0	30	20	21.62	24	0.45	89	22	26.95	30	2	-5.33
100.15.30.15.0	10	0	0	0	0.35	73	0	0.6	2	0	-0.6
100.15.30.20.0	20	19	20.78	23	0.40	80	21	26.22	29	2	-5.44

Table 4. Comparative results of TS and SDRW, maximum moves = 100,000

From Table 4, we observe that the results of SDRW are much worse than those of TS. If we compare these results with those of MCRW (Table 1), it is easy to see that SDRW gives even worse results than MCRW does. These results show that the high performance of the TS algorithm is effectively due to the tabu memory.

Table 5 gives information about the running time on a Sun SPARCstation 5 (32 RAM, 75MHz) of the TS and MCRW algorithms[2]. The second and third two columns (*time*) indicate the average running time in seconds of TS and MCRW for carrying out 100,000 *moves*.

Problem	Tabu	MCRW	Problem	Tabu	MCRW
	time	*time*		*time*	*time*
50.10.10.60.0	64	61	250.25.03.55.0	190	196
50.10.10.70.0	80	70	250.25.07.30.0	200	178
50.10.30.30.0	73	72	250.25.10.21.0	183	190
50.10.30.35.0	89	81	250.25.10.22.0	206	183
50.10.50.20.0	90	86	250.25.10.23.0	226	193
100.15.10.40.0	84	96	300.30.03.50.0	231	226
100.15.10.45.0	111	111	300.30.05.35.0	234	239
100.15.10.50.0	154	127	300.30.07.25.0	205	215
100.15.30.15.0	124	121	500.30.04.25.0	255	279
100.15.30.20.0	142	142			

Table 5. Running times of TS and MCRW for 100,000 moves

From Table 5, we observe that the running time for TS is only about 15% more important than MCRW (more for small size instances, less for large size instances) even if TS searches more larger neighborhood at each iteration. Taking into account Tables 4 and 1, we see that TS is much more efficient than MCRW.

[2] Both TS and MCRW are implemented in C++.

5 Conclusion

In this paper, we presented a basic TS algorithm for solving the maximal constraint satisfaction problem. This algorithm was tested and compared on random instances of various sizes (ranging from n/d = 50/10 to n/d = 500/30). Empirical evidence shows that the TS algorithm always finds solutions of better quality, i.e. solutions having smaller number of violated constraints.

Moreover, the TS algorithm is about 3 to 5 times faster than the MCRW algorithm to find solutions of the same quality, in terms of number of moves but also of running time.

This study shows clearly that the TS algorithm is very competitive compared with one of the best repair methods. More generally, Tabu Search has other important features such as intensification and diversification which may improve further the performance of the algorithm. Therefore, TS should be considered to be very promising for solving the MCSP.

Several points are worthy of further studies. First, we do not know if the results of this study would remain valid on instances of other classes or models of MCSP. Moreover, it will be interesting to compare Tabu Search with the most efficient complete algorithms for solving satisfiable instances of CSP.

Acknowledgments

We would like to thank the referees of this paper for their useful comments.

References

1. P. Cheeseman, B. Kanefsky and W.M. Taylor, "Where the really hard problems are", Proc. of the 12th IJCAI'90, pp163-169, 1991.
2. D.A. Clark, J. Frank, I.P. Gent, E. MacIntyre, N. Tomov, T. Walsh, "Local search and the number of solutions", Proc. of CP97, pp119-133, 1996.
3. C. Fleurent and J.A. Ferland, "Genetic and hybrid algorithms for graph coloring", to appear in G. Laporte, I. H. Osman, and P. L. Hammer (Eds.), Special Issue Annals of Operations Research, "Metaheuristics in Combinatorial Optimization".
4. E.C. Freuder and R.J. Wallace, "Partial constraint satisfaction", Artificial Intelligence, Vol.58(1-3) pp21-70, 1992.
5. F. Glover and M. Laguna, "Tabu Search", in C. R. Reeves (Ed.), Modern heuristics for combinatorial problems, Blackwell Scientific Publishing, Oxford, GB, 1993.
6. J.K. Hao and R. Dorne, "Empirical studies of heuristic local search for constraint solving", Proc. of CP-96, LNCS 1118, pp194-208, Cambridge, MA, USA, 1996.
7. P. Hensen and B. Jaumard, "Algorithms for the maximum satisfiability problem", Computing Vol.44, pp279-303, 1990.
8. A. Hertz and D. de Werra, "Using Tabu search techniques for graph coloring". Computing Vol.39, pp345-351, 1987.
9. T. Hogg, B.A. Huberman and C.P. Williams, Artificial Intelligence, Special Issue on the Phase Transition and Complexity. Vol 82, 1996.
10. D.S. Johnson, C.H. Papadimitriou and M. Yannakakis, "How easy is local search?" Journal of Computer and System Sciences, Vol.37(1), pp79-100, Aug. 1988.

11. S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, "Optimization by simulated annealing",Science No.220, pp671-680, 1983.
12. J. Larrosa and P. Meseguer, "Optimization-based heuristics for maximal constraint satisfaction", Proc. of CP-95, pp190-194, Cassis, France, 1995.
13. A.K. Mackworth, "Constraint satisfaction", in S.C. Shapiro (Ed.) Encyclopedia on Artificial Intelligence, John Wiley & Sons, NY, 1987.
14. S. Minton, M.D. Johnston and P. Laird, "Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems", Artificial Intelligence, Vol.58(1-3), pp161-206, 1992.
15. P. Morris, "The Breakout method for escaping from local minima", Proc. of AAAI-93, pp40-45, 1993.
16. C.H. Papadimitriou and K. Steiglitz, "Combinatorial optimization - algorithms and complexity", Prentice Hall, 1982.
17. B. Selman and H.Kautz, "Domain-independent extensions to GSAT: solving large structured satisfiability problems", Proc. of IJCAI-93, Chambery, France, 1993.
18. B.M. Smith, "Phase transition and the mushy region in constraint satisfaction problems", Proc. of ECAI94, pp100-104, 1994.
19. E. Tsang, "Foundations of constraint satisfaction", Academic Press, 1993.
20. R.J. Wallace, "Enhancements of branch and bound methods for the maximal constraint satisfaction problem", Proc. of AAAI-96, pp188-196, Portland, Oregon, USA, 1996.
21. R.J. Wallace, "Analysis of heuristics methods for partial constraint satisfaction problems", Proc. of CP-96, LNCS 1118, pp308-322, Cambridge, MA, USA, 1996.
22. N. Yugami, Y. Ohta and H. Hara, "Improving repair-based constraint satisfaction methods by value propagation", Proc. of AAAI-94, pp344-349, Seattle, WA, 1994.