A Local Search Algorithm for a SAT Representation of Scheduling Problems

Marco Antonio Cruz-Chávez[1] and Rafael Rivera-López[2]

[1] CIICAp, Autonomous University of Morelos State
Av. Universidad 1001, Chamilpa, 62209, Cuernavaca, Morelos, México
mcruz@uaem.mx
[2] Technological Institute of Veracruz
Miguel Ángel de Quevedo 2779, Formando Hogar, 91860 Veracruz, Veracruz, México
rrivera@itver.edu.mx

Abstract. This paper presents the application of a local search algorithm for a logical representation of the Job Shop Scheduling Problem (JSSP). This logical representation represents the JSSP transformed as a satisfiability problem (SAT). The proposed algorithm uses a local search in a wide neighborhood. This algorithm, called Walk Wide Search - SAT, is a variant of the WalkSAT algorithm. This search is possible because the included tabu list prevents an excessive number of repetitions of movements during the search process. This paper describes the algorithm and compares results of Walk Wide Search - SAT to WalkSAT.

Keywords: Job shop, satisfiability, SAT formula, disjunctive graph, Reduced SAT Codification of JSSP.

1 Introduction

The complexity theory groups computational problems according to the inherent difficulty of solving them. This classification is used to classify decision problems [1], which require a yes/no answer in order to obtain their solution. The best-known problem in this classification is the satisfiability problem, which is classified as NP-complete. The objective of the satisfiability problem is to confirm or deny the existence of an assignment of truth-values for the literals of a logic formula (SAT formula) which make the formula true. A SAT formula is usually written in its conjunctive normal form (CNF) that has the following three features: (1) a conjunction C of clauses C_i, i.e. $C = C_1 \wedge C_2,\ldots, \wedge C_n$, (2) each clause C_i is a disjunction of literals $X_i \vee \ldots \vee X_k$, (3) each literal X_j is a Boolean variable (negated or not).

Some decision problems could be defined like discrete optimization problems. An instance of a discrete optimization problem is defined by the function $c : F \rightarrow R$, where F is the finite set of solutions that defines the problem instance, R is the set of discrete values that define each solution in F, and c is the objective function. In an instance of a discrete optimization problem, it is necessary to find the solution $f \in F$ for which $c(f) \leq c(y), \quad \forall y \in F$.

O. Gervasi and M. Gavrilova (Eds.): ICCSA 2007, LNCS 4707, Part III, pp. 697–709, 2007.

The Job Shop Scheduling Problem (JSSP) [2] is one of the most difficult problems within the NP-complete classification [1]. As a discrete optimization problem, it is an NP-hard practical problem [3] found in the area of manufacturing. For these two reasons, JSSP is a problem of great interest to the scientific community.

The JSSP is frequently treated as a discrete optimization problem; in this paper, the JSSP is treated as a decision problem based on its representation in the form of a satisfiability problem, in order to work with a proposed local search algorithm. By working in this way, the proposed local search algorithm looks for the satisfiability of a SAT formula that represents JSSP. The satisfiability of the SAT formula provides a feasible schedule of the JSSP. With this schedule, the proposed local search algorithm for the satisfiability problem could be applied in the search for solutions in a discrete optimization problem, as JSSP is generally treated.

Various methods have been proposed for manipulate the JSSP using several models. Two of the most commonly used models are disjunctive graphs [4] and integer programming [5]. The methods used with these models can be divided into two groups, local search methods and optimization methods.

Local search methods are iterative procedures that allow movement from one solution in F to another. These methods search for a local optimum in the solutions space of the problem through the use of a neighborhood function, which is defined as follows: Given a feasible point $f \in F$ in an instance of a problem, a neighborhood of f is defined as a set $N(f)$ of feasible points near f. The set $N(f)$ called the neighborhood f, indicates that each solution $f' \in N(f)$ can be reached directly from f in one step. In accordance with this the neighborhood is defined by the function $N: F \rightarrow 2^F$. Some of the most well-known local search methods are Hill-climbing, Steepest-descent, Iterative improvement, GSAT and WalkSAT.

Optimization methods try to find the best solution evaluating the objective function of the problem in order to find their maximum or minimum value. Generally, optimization methods use local search methods within their procedure. The methods based on Branch and Bound [6], Simulated Annealing [7], Genetic Algorithms (GA) [8] and Shifting Bottleneck [9], [10] are optimization methods, which try to determine the best solution according to definite criteria of the objective function. At the moment the line has been taken of working with hybrid algorithms, which combine in their procedure global and local searches. El-Mihoub et al. [11] and Shannon [12], present a revision of hybrid algorithms that combine the power of the GA and the local search. In this revision, it is affirmed that a GA could quickly locate the region which contains the global optimum, but it takes a relatively long time to locate the local optimum in the region of convergence. The combination of the GA and a local search method could accelerate the search in order to locate the global optimum. All of these works demonstrate that a GA combined with local search improves greatly their efficiency in a wide variety of problems. This type of GA combined with local search methods is known as the Memetic Algorithm [13].

Because optimization methods incorporate the use of local search methods into their procedures, local search methods that provide feasible solutions in an efficient manner can be very useful in providing a good starting point for the optimization methods [14].

Most of the methods used to find solutions to the satisfiability problem [15], [16], [17] are local search methods. This paper presents an algorithm of local search called

Walk Wide Search - SAT (WWS-SAT) for SAT problems, which can be applied to the JSSP encoded as satisfiability (SAT) using a codification format called reduced SAT [18]. The algorithm proceeds first to generate a propositional formula in conjunctive normal form called this SAT formula that represents the JSSP. Next, the algorithm looks for an assignment of truth-values for the variables in the SAT formula which satisfy the formula. If the satisfiability of the SAT formula is obtained, it implies that the result is a feasible schedule of the problem. In general, all the possible assignments of truth-values that satisfy a SAT formula form the set of feasible schedules of a JSSP instance. The paper presents experimental results of WWS-SAT which present it like a local search method very efficient and that one could apply as the base on the optimization methods in order to make them more efficient.

This paper is organized in five sections. Section one is the introduction. Section two describes the model of the disjunctive graph of JSSP. Section three presents the process of SAT codification for JSSP [18], which is explained briefly. Section four presents the proposed algorithm and an example of the assignment of truth-values to a SAT formula derived for an instance of JSSP. Section five presents the experimental results. Section six presents the conclusions and future work.

2 The Disjunctive Graph Model

The disjunctive graph model represents JSSP as a disjunctive graph G [4], [5], which is a 3-tuple where G = (N, E, A). N is the set of operations which includes two fictitious operations, the start and the end of the JSSP. These operations are represented in the nodes of G. E and A are two sets, edges and arcs (disjunctive and conjunctive) respectively. The set of edges (resource capacity constraints) is formed of subsets of these edges, each one of these subsets represents a machine M_i, E = $\{M_1, M_2, M_3, .., M_m\}$. The set of arcs (precedence constraints) is formed of subsets of these arcs, each one of these subsets represents a job, J_i, A = $\{J_1, J_2, J_3, .., J_n\}$.

Figure 1 represents the JSSP of two jobs and two machines (2x2). The problem has four operations, numbered 1 to 4, which are represented by the nodes of the graph. The start and the end operations are I and * respectively. The processing time of each operation is noted beside the corresponding node. Job 1 consists of operations 1 and 2, while job 2 consists of operations 3 and 4. P_1 and P_2 are the precedence constraints and they connect each pair of operations that belong to the same job. P1 means that in job 1, operation 2 cannot begin before operation 1 finishes. Likewise, P_2 prohibits the starting of job 4 until job 3 is finished. Machine M_1 executes the operations 1 and 4, while machine M_2 executes operations 2 and 3. R_1 and R_2 are the resource capacity constraints and they connect each pair of operations that are executed by the same machine. R_1 means that for M_1, operation 1 can be executed before 4 or 4 can be executed before 1, but the two operations cannot be simultaneously executed. Similarly, R_2 means that M_2 can carry out operations 2 and 3 in any order, but not simultaneously. In order to find a feasible schedule for this model, it is necessary to find the direction of each edge R, such that the formed schedule does not contain cycles.

3 Reduced SAT Codification

The reduced SAT codification of JSSP [18] is a SAT formula that contains a smaller number of clauses than other proposed SAT encodings for problems of scheduling [16], [17].

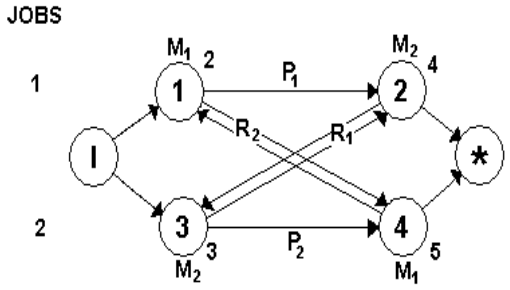

Fig. 1. JSSP of 2x2 represented by a disjunctive

Each clause in the reduced SAT codification is formed using only the restrictions of R and not involving the restrictions of P (Figure 1) due to the use of the latest starting time (LST) as the start time ***s*** of each operation [18]. The restrictions R, are defined as $s_i + p_i \leq s_j \vee s_j + p_j \leq s_i$ [5] for a pair of operations *i, j*, where *s* and *p* are the start time and processing time of the operation. The obtained clauses can be seen in Table 1; they are the result of mapping the resource capacity constraints as a set of variables that take values only of true or false. For each R, two clauses are obtained. In Table 1, it is shown that for each set of variables $sa_{k,t}$ where *k* represents any operation, the operation *k* begins at or after the time *t*. In addition, $sa_{j,t+pi}$ means that the operation *j* begins at or after the time $t+p_i$. $Pr_{i,j}$ means that the operation *i* precedes *j* if the truth-value that it possesses is true. If however the truth-value is false then *j* precedes *i*.

Table 1. Set of clauses that compose the SAT formula

Clauses of *R*	CNF	Interpretation
$sa_{i,t} \wedge pr_{i,j} \rightarrow sa_{j,t+pi}$	$\sim sa_{i,t} \vee \sim pr_{i,j} \vee sa_{j,t+pi}$	If *i* Starts at or after *t* and *j* follows *i* then *j* cannot start until *i* is finished.
$sa_{j,t} \wedge pr_{j,i} \rightarrow sa_{i,t+pj}$	$\sim sa_{j,t} \vee \sim pr_{j,i} \vee sa_{i,t+pj}$	If *j* Starts at or after *t* and *i* follows *j* then *i* cannot start until *j* is finished

In order to find a feasible schedule for JSSP, it is only necessary to find a truth assignment that satisfy the SAT formula which is formed by the set of clauses defined in Table 1. In order to be able to evaluate the set of variables sa_k and $sa_{j,t+pi}$, it is necessary to know the start time *t* of each operation. These times are calculated using the digraph that is obtained from the disjunctive graph that represents the JSSP. The digraph represents a sequence of execution of operations, which is generated with the truth-values of the set of variables $Pr_{i,j}$ assigned in random form. The time *t* (LST) of each operation *i* is equal to the critical path (longest path) [18] generated in the

digraph between the operation i and the operation initial I. When there is an assignment of truth-values that satisfy the SAT formula, a feasible schedule is obtained as a result.

4 The Local Search Algorithm

An algorithm of local search always requires the assignment of truth-values to the variables that compose the SAT formula, which is formed by the clauses in Table 1. In order to reduce the solution space where the search will be carried out, a set of control variables is designated. By designating a set of control variables, it is possible to determine the truth-values of the remaining variables. The variables used as control variables are the variables $pr_{i,j}$ (Table 1), because a finite number of sequences of pairs of operations in a problem exists. In contrast, by using the variables *sa* as the control variables, the situation becomes more complicated because the variables *sa* involve start times, of which there are an infinite number for each operation. By designating the variables $pr_{i,j}$ as control variables, and assigning truth-values to these variables, the formation of a sequence of the execution of operations in the problem results. With the defined sequence, a digraph of the disjunctive graph is generated which allows the LST for each operation to be calculated. Because the LST is calculated in this way, the first variable ($sa_{i,t}$) in each clause will always be true and its negation false ($\tilde{}sa_{i,t}$). The previous observation leaves the SAT formula as a formula 2SAT, that is, a formula where each clause contains only two variables ($\sim pr_{i,j} \vee sa_{j,t+pi}$). The LST and the number of clauses are obtained by simplifying the digraph. The simplification involves using only the Hamilton routes [19] that are generated for each machine. The calculation of the LST is then one of linear order [10], [19] and the number of clauses decreases because the number of edges R is decreased.

One could see that the local search algorithm proposed in this paper is an hybrid algorithm because it requires of an assignment of truth-values to the control variables ($pr_{i,j}$) and also LST must be calculated in order to be able to evaluate the remaining variables ($sa_{j,t+pi}$) of the SAT formula, this last is not common in algorithms for satisfiability. After obtaining the complete assignment of truth-values for the SAT formula, the satisfiability of the SAT formula is evaluated in a following step.

4.1 Example of Assignment of Truth-Values to the SAT Formula

Next is the presentation of an example of the assignment of truth-values to the variables of control in the SAT formula. This example is the 2x2 JSSP represented by the disjunctive graph found in Figure 1. The total number of clauses involved is determined by equation (1) and is equal to four clauses, where *nmaq* is the number of existing machines.

$$Clauses = 2\ nmaq\ (nmaq\text{-}1) \tag{1}$$

$$Variables\ of\ Control = nmaq\ (nmaq\text{-}1) \tag{2}$$

The number of variables of control (*pr*) is determined by equation (2) and is equal to two, one variable for each pair of operations. The possible assignments of

Table 2. Assignment of truth-values for the SAT formula of a JSSP of 2x2

Variables		Truth-values CASES			
C	D	1	2	3	4
$P_{1,4}$		*F*	*F*	*T*	*T*
	$P_{4,1}$	*T*	*T*	*F*	*F*
$P_{3,2}$		*F*	*T*	*F*	*T*
	$P_{2,3}$	*T*	*F*	*T*	*F*

truth-values are obtained using the relationship (*nmaq!*)*nmaq*. In this example, the possible number of assignments of truth-values is four, as shown in Table 2.

C are the variables of control and D are variables whose value depends on C. The resulting SAT formula is $\left(\neg \mathrm{Pr}_{1,4} \vee Sa_{4,t_1+p_1}\right) \wedge \left(\neg \mathrm{Pr}_{4,1} \vee Sa_{1,t_4+p_4}\right) \wedge \left(\neg \mathrm{Pr}_{3,2} \vee Sa_{2,t_3+p_3}\right) \wedge \left(\neg \mathrm{Pr}_{2,3} \vee Sa_{3,t_2+p_2}\right)$. As could be seen in this SAT formula, the number of Boolean variables $Sa_{k,tl+pl}$ is equal to the number of generated clauses. That means that the existence of a variable $Sa_{k,tl+pl}$ is observed for each existent operation in the problem, where *k= 1, 2, ... , nxm*, *n* is the number of jobs, *m* is the number of machines and *nxm* is the number of operations for a symmetrical problem of JSSP. The characteristic of generation of a small amount of Boolean variables in order to represent an instance of JSSP is characteristic of the reduced SAT codification [18], in this codification the variables $Sa_{k,tl+pl}$ are not generated in each unit of time $tl + pl$ as in other proposed SAT encodings for JSSP [16], [17]. Instead, variables $Sa_{k,tl+pl}$ only exist as defined by the units of time LST calculated in the operations of the problem. In order to evaluate a SAT formula only an LST for each operation is calculated.

When the four cases of assignment are proven in the SAT formula, it is found that in the first case, satisfiability of the formula is not obtained, but in the following three cases it is. A closer examination of case 1 and case 2 follows.

Case 1

The truth-values from Table 2 are assigned to the variables of control (C) and dependents (D). Then the LST of each operation can be calculated using the digraph generated by Figure 1 and Table 2. This digraph can be seen along with the LST of each operation in Figure 2. The start times of each operation grow to an infinite value because the operations in the digraph in Figure 2 belong to global cycles. Each operation is assigned a start time with a very high value, in this case 1000 is the designated start time. Substituting the times in each variable of the SAT formula results in the following: $\left(\neg P_{1,4} \vee Sa_{4,1002}\right) \wedge \left(\neg P_{4,1} \vee Sa_{1,1005}\right) \wedge \left(\neg P_{3,2} \vee Sa_{2,1003}\right) \wedge \left(\neg P_{2,3} \vee Sa_{3,1004}\right)$. Through the evaluation of each variable using Table 2 and Figure 2, it can be seen that the satisfiability of the formula is not obtained, that is (T $\vee$ F) $\wedge$ (F $\vee$ F) $\wedge$ (T $\vee$ F) $\wedge$ (F $\vee$ F).

Evaluation of Boolean variables $Sa_{k,tl+pl}$. In this case, all variable $Sa_{k,tl+pl}$ take the truth-value of false because the start time obtained by LST in each operation *k* grown to an infinite value.

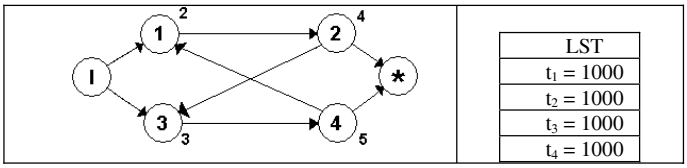

Fig. 2. Digraph and the assignment of LST in Case 1

Case 2

The truth-values from Table 2 are assigned as the control variables (C) and dependents (D). Then the LST of each operation can be calculated using the digraph generated by Figure 1. This can be seen along with the LST of each operation in Figure 3. By substituting the times in each variable of the SAT formula, the result is $\left(\neg P_{1,4} \vee Sa_{4,10}\right) \wedge \left(\neg P_{4,1} \vee Sa_{1,8}\right) \wedge \left(\neg P_{3,2} \vee Sa_{2,3}\right) \wedge \left(\neg P_{2,3} \vee Sa_{3,14}\right)$. Through the evaluation of each variable using Table 2 and Figure 3, it can be seen that the satisfiability of the formula is obtained, that is (T $\vee$ F) $\wedge$ (F $\vee$ T) $\wedge$ (F $\vee$ T) $\wedge$ (T $\vee$ F).

Evaluation of Boolean variables $Sa_{k,tl+pl}$. In this case, the operation does not have a start time that grown to an infinite value (see Figure 3).

Clause 1. The Boolean variable $P_{1,4}$ takes the truth-value of false, this means that operation 1 does not precede operation 4 (see Figure 3), therefore, operation 4 should start before and end before or at the same time that operation 1 starts. This is not true because in clause 1 it can be observed that operation 4 starts in 10 and ends in 15 (processing time is added, see Figure 3) and operation 1 stars in 8 (see Figure 3) Therefore, the Boolean variable $Sa_{4,10}$ takes the truth-value of false.

Clause 2. The Boolean variable $P_{4,1}$ takes the truth-value of true, this means that operation 4 precedes the operation 1 (see Figure 3), therefore, operation 4 should start before and end before or at the same time that operation 1 starts. This is true because operation 4 starts in 3 and ends in 8 (processing time is added, see Figure 3) and in clause 2 it can be observed that operation 1 starts in 8, therefore the Boolean variable $Sa_{1,8}$ takes the truth-value of true.

Clause 3. The Boolean variable $P_{3,2}$ takes the truth-value of true, this means that operation 3 precedes operation 2 (see Figure 3), therefore, operation 3 should start before and end before or at the same time that operation 2 starts. This is true because operation 3 starts in 0 and ends in 3 (processing time is added, see Figure 3) and in clause 3 it can be observed that operation 2 starts in 3. Therefore, the Boolean variable $Sa_{2,3}$ takes the truth-value of true.

Clause 4. The Boolean variable $P_{2,3}$ takes the truth-value of false, this means that operation 2 does not precede the operation 3 (see Figure 3). Therefore, operation 3 should start before and end before or at the same time that operation 2 stars. This is not true because in clause 4 it can be observed that operation 3 starts in 14 and ends in 17 (processing time is added, see Figure 3) and operation 2 begins in 10 (see Figure 3). Therefore, the Boolean variable $Sa_{3,14}$ takes the truth-value of false.

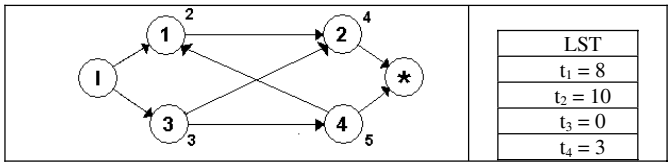

Fig. 3. Fig. 3 Digraph and the assignment of LST in Case 2

4.2 Walk Wide Search-SAT Algorithm

The local search algorithm, which is presented here and called Walk Wide Search-SAT (WWS-SAT), is based on the WalkSAT algorithm [20]. WWS-SAT is an algorithm that looks for the satisfiability of a SAT formula. This algorithm was developed for practical problems of Job Shop Scheduling. In contrast to theoretical problems [21], which are commonly used with the problem of satisfiability, the SAT formula generated from JSSP presents different characteristics. One such characteristic is that repeated variables do not exist. These differences cause changes in the behavior of the algorithms that are frequently used in the SAT area, which results in their being less efficient [16].

The WalkSAT algorithm begins with a randomly generated assignment of truth-values for the SAT formula. This assignment of truth-values is defined as Assignment-A. With a probability P, the search focuses on variables of clauses not satisfied. With a probability 1-P, the search focuses on variables of clauses taken randomly and chooses a change in the assignment of the truth-values of the variables. By changing the assignment of truth-values, the number of clauses that are not satisfied is diminished. The WalkSAT algorithm requires three parameters: (1) The set of clauses that generate the SAT formula to be evaluated, (2) the values of MAX-FLIPS that determine the number of exchanges that will be attempted, and (3) the value of MAX-TRIES that determine the number of times the search will restart before finishing. The algorithm presented here, WWS-SAT, can be described as a walk wide search because search in a wide neighborhood and carries out more than one movement in the neighborhood. In Table 3, it can be seen that WWS-SAT is a modification of WalkSAT. With a probability P, WWS-SAT focuses the search on variables of clauses not satisfied. WWS-SAT also generates a list of all the clauses not satisfied and includes taboo movements as well. This list is defined as List-C. WWS-SAT eliminates from list the clauses in which there are repeated operations of the control variables pr_{ij}.

Table 3. WalkWideSearch-SAT algorithm

```
WWS-SAT(FormulaSAT){
 For i=1 to MAX-TRIES{
  A = a randomly generated truth assignment;
   For j = 1 to MAX-FLIPS{
    If A is a solution return;
     Else{
      C = Choosing the total of unsatisfied clauses
          where they don't repeat operations in pr;
          With probability P
             Flip variables pr_{i,j} in C;
          Otherwise (with probability 1-P)
             Flip a variable in an unsatisfied
               clause of C;
     }
   }
 }
 return failure;
}
```

For example, if there are three different clauses which are not satisfied, involving the variables $pr_{3,4}$, $pr_{4,5}$ and $pr_{5,6}$, with one variable *pr* in each clause, it can be observed that operation 4 repeats in the first and second variable and operation 5 in the second and third variable. If one of the clauses with variables that contain repeated operations is eliminated, the result could be $pr_{3,4}$ and $pr_{5,6}$. This avoids falling into continuous cycles within a sequence of operations. This process enables the satisfiability of a SAT formula to be found in a shorter amount of time because the satisfiability of a SAT formula is a feasible schedule, which is a sequence of operations that does not contain any cycles. Once the List-C has been diminished by the elimination of clauses, the assignment of truth-values is changed for the control variables in remaining clauses of the List-C. It can then be seen whether assignment A is a solution or not. Alternately, with a probability 1-P, an unsatisfied clause can be taken randomly and the assignment of truth-values for the control variables can be changed. Then the Assignment-A can be tried out as a solution.

5 Experimental Results

In order to carry out the tests, a personal computer with a processor of 1.0 GHz and 1 GB in RAM was used. The operative system was Windows Millennium and the programming language was Visual C++ 6.0.

In order to prove the efficiency of the WWS-SAT algorithm, we carried out several comparative tests between it and the WalkSAT algorithm. The reported time in this paper is the time that takes the algorithm to satisfy the SAT formula, it does not include the time spent loading the program in memory or generating the SAT formula,

because the time to generate the SAT formula is very short because a reduced codification with few clauses is always generated [18]. The problems that were used in the tests can be found in the OR library [22]. They include: FT06 of 6x6, FT10 of 10x10, LA16 to LA20 of 10x10, ORB1 to ORB10 of 10x10, and LA36 of 15x15. For both algorithms, the probability used was P = 1. This was necessary because for a smaller probability, the results become much less efficient. The decrease in efficiency can be seen in Figure 4 where the graph of the problem FT06 using WalkSAT is presented. For each value of P, the number of flips in an average of 700 tests was obtained. It can be observed that as the value of P decreases, a higher number of flips are necessary in order to find the satisfiability of the SAT formula. As a consequence of the high number of flips, the time increases. The same behavior can be observed with WWS-SAT, so it was decided to use P= 1 for both algorithms when conducting the comparative tests. The results for WalkSAT and WWS-SAT used with problems of different sizes are shown in Tables 4 and 5 respectively. The presented results are averages of 2000 tests for FT06, 2000 for FT10, and 2000 for LA36. The percentage of achievements by each algorithm has a maximum of 100%. The number of flips, which are changes in the assignment of truth-values for a variable, is smaller in WWS-SAT than in WalkSAT. For FT06 the number of flips is 74% less, for FT10 it is 91% less, and for LA36 it is 6% less. The time to obtain the satisfiability of the SAT formula is also less for WWS-SAT. For FT06 the time is 81% less, for FT10 it is 45% less, and for LA36 it is 21% less. This indicates that in problems of varying sizes the efficiency of WWS-SAT is much better than that of WalkSAT. In addition, the efficiency of the generation of flips using WWS-SAT is better than that of WalkSAT.

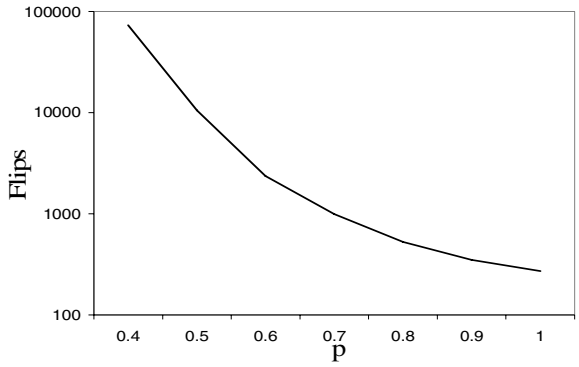

Fig. 4. Problem FT06. Flips vs. P. WalkSAT

Table 4. Experimental results in problems of several sizes for WalkSAT

WalkSAT		
JSSP	Success rate	Flips
FT06	100	00272
FT10	100	03853
LA36	100	47858

Table 5. Experimental results in problems of several sizes for WWS-SAT

WWS-SAT		
JSSP	Success rate	Flips
FT06	100	00072
FT10	100	00358
LA36	100	45066

Table 6 shows the results for the eighteen problems of 10x10 presented in the OR library. The results were obtained by running an average of 50 tests per problem. From Table 6, it can be noted that for the problems Orb3 and Orb8, it is difficult to satisfy the SAT formula with the WalkSAT algorithm because it takes an average of 0.017 and 0.01933 seconds respectively. That is not the case with WWS-SAT because the same problems only take 0.00124 and 0.00112 seconds to satisfy the SAT formula. It can also be observed that in most of the problems, WWS-SAT requires a smaller number of flips.

Table 6. Experimental results of the scheduling problems of 10x10, in OR Library

JSSP 10x10	WalkSAT		WWS-SAT	
	Flips	Time (s)	Flips	Time (s)
FT10	3968	0.00388	350	0.00176
Abz5	5495	0.00170	5071	0.00094
Abz6	5471	0.00163	5304	0.00096
La16	5113	0.00167	5117	0.00118
La17	5284	0.00160	5354	0.00116
La18	5652	0.00190	5212	0.00136
La19	5416	0.00157	5521	0.00126
La20	5486	0.00170	4741	0.00086
0rb1	4209	0.00476	366	0.00214
0rb2	5006	0.00167	5003	0.00136
0rb3	3259	0.01700	332	0.00124
0rb4	4521	0.00207	4305	0.00162
0rb5	4630	0.00280	4244	0.00190
0rb6	4321	0.00390	353	0.00207
0rb7	12631	0.00508	11819	0.00407
0rb8	3425	0.01933	341	0.00112
0rb9	4474	0.00200	4526	0.00136
0rb10	4551	0.00280	4547	0.00228

The exceptions are for LA16, LA19 and ORB9, which require fewer flips for WalkSAT. Even in these exceptions, the number of flips used by both WalkSAT and WWS-SAT is very similar but WWS-SAT wins in most of the problems, see FT10, Orb1, Orb3, Orb6 and Orb8 where the difference in flips is bigger. It should be noted that in all cases, WWS-SAT requires less time. By taking an average of all the problems listed in Table 6, it can be seen that WWS-SAT is more efficient because it generates 19% less flips and satisfies the SAT formula 63% more quickly than WalkSAT.

6 Conclusions

It has been demonstrated here that for tests carried out in practical problems like JSSP, WWS-SAT is superior to WalkSAT. For the SAT formula in JSSP, it is most convenient use a P = 1, due to the characteristics of the formula. This is because each pair of variables present in each clause does not repeat in any another clause. The use of a taboo list in WWS-SAT avoids returning to previous movements and avoids falling continually in sequences of operations that form cycles, consequently, allowing WWS-SAT to be applied to a bigger neighborhood. This larger neighborhood accelerates the acquisition of an assignment of truth-values that satisfy the SAT formula. For several problems of the same size, it is demonstrated that the use of WWS-SAT is uniform and does not change, as does WalkSAT. This constancy is achieved by the walk wide applied in WWS-SAT's local search. Instead of making a change in the assignment of a single variable, several changes are conducted in several variables that do not belong to the taboo list. These changes prevent stagnation in the local search, which happens in the search of WalkSAT because WalkSAT changes the assignment of one truth-value for one variable in one instant of time. The experimental results present to WWS-SAT like a local search method very efficient, because it find feasible solutions in a very simple form. This presents to WWS-SAT very attractive for their use in the optimization methods.

As seen when using the WWS-SAT algorithm (Section 4), the assignment of truth-values to the pr_{ij} variables is random. This allows for better distribution of the feasible schedules generated; they are located throughout the entire solution space of a JSSP. This characteristic could be of great interest for those who use Genetic Algorithms, because the better distribution of populations that are generated in the solution space could improve the search in a JSSP.

7 Future Work

WWS-SAT will be implemented with the algorithms of Simulated Annealing and GA in order to evaluate the efficacy of this local search method. In these optimization methods the Makespan as an objective function will be used in order to evaluate the efficacy of WWS-SAT. Efficacy will be evaluated as the ease with which solutions near the global optimum are found for the objective function in question.

References

1. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity, p. 496. Dover Publications, Mineola (1998)
2. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of Flow shop and Job shop Scheduling. Mathematics of Operations Research 1(2), 117–129 (1976)
3. Brucker, P.B.: Scheduling algorithms, 5th edn. Springer, Heidelberg (2007)
4. Balas, E.: Machine Sequencing via Disjunctive Graphs. An Implicit Enumeration Algorithm, Operations Research 17, 941–957 (1969)
5. Conway, R.W., Maxwell, W.L., Miller, L.W.: Theory of Scheduling. Addison Wesley, Massachusetts (1967)

6. Jalilvand, A., Khanmohammadi, S., Shabaninia, F.: Scheduling of sequence-dependant jobs on parallel multiprocessor systems using a branch and bound-based Petri net, Emerging Technologies. In: Proc of the IEEE, pp. 334–339 (September 17-18, 2005) ISBN: 0-7803-9247-7
7. Cruz-Chávez, M.A., Frausto-Solís, J., Zavala-Díaz, J.C., Sanvicente-Sánchez, H., Cruz-Rosales, M.H.: A Simulated Annealing Algorithm with Cooperative Processes for Scheduling Problems. LNCS. Springer, Heidelberg (to appear, 2006) ISSN: 0302-9743
8. Zalzala, P.J., Flemming.: Genetic algorithms in engineering systems, in A.M.S. Inst. of Electrical Engineers (1997)
9. Defu, Z., Tangqiu, Li., Shaozi, Li.: An improved shifting bottleneck algorithm for job shop scheduling problem. In: Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design. vol. 2, pp. 1112–1116, 24-26 (May 2005)
10. Schutten, M.J.: Practical job shop scheduling. In Annals of Operations Research 83, 161–177 (1988)
11. El-Mihoub, T.A., Hopgood, A.A., Nolle, L., Battersby, A.: Hybrid Genetic Algorithms: A Review, Engineering Letters (2006) ISSN: 1816-0948, 13:2, EL_13_2_11
12. Shannon Land, M.W.: Evolutionary Algorithms with Local Search for Combinatorial Optimization, Ph.D. Thesis, University of California, San Diego, p. 169 (1998)
13. Krasnogor, N., Smith, J.: A Memetic Algorithm With Self-adaptive Local Search: TSP as a case study. In: Whitley, Goldberg, Cantu-Paz, Spector, Parmee, Beyer (eds.) Proceedings of GECCO 2000, pp. 987–994. Morgan Kaufmann, San Francisco (2000)
14. Cruz-Chávez, M.A., Frausto-Solís, J.: Simulated Annealing with Restart to Job Shop Scheduling Problem Using Upper Bounds. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 860–865. Springer, Heidelberg (2004)
15. Ullman, J.D.: NP-complete scheduling problems. Journal of Computer System Sciences 10, 384–393 (1975)
16. Crawford, J.M., Baker, A.B.: Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In: Proc. of the 12th National Conf. on Artificial Intelligence, Austin, TX, pp. 1092–1098 (1994)
17. Memik, S.O., Fallah, F.: Accelerated SAT-based scheduling of control/data flow graphs Computer Design: VLSI in Computers and Processors, pp. 395–400, Proc IEEE (September 16-18, 2002)
18. Frausto-Solís, J., Cruz-Chávez, M.A.: A Reduced Codification for the Logical Representation of Job Shop Scheduling Problems. In: Laganà, A., Gavrilova, M., Kumar, V., Mun, Y., Tan, C.J.K., Gervasi, O. (eds.) ICCSA 2004. LNCS, vol. 3046, pp. 553–562. Springer, Heidelberg (2004)
19. Garey, M.R., Johnson, D.S.: Computers and intractability: a guide to the theory of NP-completeness, p. 340. W.H. Freeman and Company, New York (1991)
20. Selman, B., Kautz, II, A.: Local search strategies for satisfiability testing. In: Procceding DIMACS Workshop on Maximum Clique, Graph Coloring and Satisfiability (1993)
21. Hoos, H.H., Stützle, T.: SATLIB: An Online Resource for Research on SAT. In: Gent, I.P., Maaren, H.v., Walsh, T. (eds.), pp. 283–292. IOS Press, Amsterdam (2000) 2006, SATLIB is available online at www.satlib.org
22. Beasley, J.E.: OR Library, Imperial College, Management School, Last Update (October 2005) (2007), http://people.brunel.ac.uk/~mastjjb/jeb/info.html