

# An Improved Integer Local Search for Complex Scheduling Problems

Weixiong Zhang and Xiaotao Zhang

Department of Computer Science and Engineering

Washington University in St. Louis

St. Louis, MO 63130, USA

Email: zhang@cse.wustl.edu

## Abstract

We consider complex scheduling problems that can be captured as optimization under hard and soft constraints. The objective of such an optimization problem is to satisfy as many hard constraints as possible and meanwhile to minimize a penalty function determined by the unsatisfied soft constraints. We present an efficient local search algorithm for these problems which improves upon Wsat(oip), a Walksat-based local search algorithm for overconstrained problems represented in integer programs. We introduce three techniques to the Wsat(oip) algorithm to extend its capability and improve its performance: backbone guided biased moves to drive the search to the regions in search space where high-quality and optimal solutions reside; sampling-based aspiration search to reduce search cost and make anytime solutions available over the course of the search; and dynamic parameter tuning to dynamically adjust the key parameters of the algorithm to make it robust and flexible for various applications. Our experimental results on large-scale crew scheduling, basketball tournament scheduling and progressive party scheduling show that the new improved algorithm can find better solutions with less computation than Wsat(oip).

## 1 Introduction and Overview

The recent advances in the research of Boolean satisfiability (SAT) have provided great insights into the problem, such as phase transitions and backbones [14; 15; 22], and have developed efficient algorithms for solving SAT, represented by the widely applied Walksat local search algorithm [13; 17] and its variants [7; 13]. The success of Walksat has also led to the paradigm of formulating and solving complex planning and scheduling problems as SAT problems [9; 11; 10]. Under this paradigm, a complex problem is encoded as a SAT problem, a solution to the SAT problem is found by applying an algorithm for SAT, and finally the solution is mapped back to the original problem. This SAT-based paradigm has been shown successful for some complex problems in real applications. For example, Blackbox

is one of the most competitive methods for planning [10; 11], which was developed under the SAT-based paradigm by applying SAT encoding and SAT algorithms.

Many constraints in real-world applications, however, are complex and may not be easily and efficiently encoded as clauses. More useful and general constraint formulations are integer linear programs (ILP) [6; 20], which allow integer variables and complex constraints, and subsume pseudo Boolean formulae with variables taking values 0 or 1 [5; 19; 20]. ILPs and pseudo Boolean formulae have been well applied to planning and scheduling problems [1; 12; 20].

Wsat(oip) [20] is an extension to the Walksat algorithm for handling overconstrained integer programs (OIPs) that involve hard and soft constraints. Here a hard constraint is one that needs to be satisfied, and a soft constraint is one that may be violated but incurs a penalty if not satisfied. The objective of such a problem is to satisfy all hard constraints, if possible, or as many hard constraints as possible when overconstrained, while minimizing a penalty function. Wsat(oip) has been shown effective on large constraint optimization and scheduling problems [12; 19; 20].

Inherited from the Walksat algorithm, Wsat(oip) is a local search algorithm that makes stochastic local perturbations to the current assignment of all variables in searching for progressively better solutions [20]. A noticeable characteristic of Walksat and Wsat(oip) is that whenever multiple choices exist, a *uniformly random* choice will be made. For example, when an unsatisfied clause is to be selected from a set of unsatisfied clauses, each qualified candidate is given equal chance to be picked. Likewise, the variable whose value is to be changed to next is chosen, uniformly randomly, from multiple candidates. Such uniform random moves are ineffective when there exist large “plateau” regions in search space, and the problem is exacerbated in OIPs and Wsat(oip) when “plateau” regions become larger due to larger domains of integer variables.

Motivated to solve complex, real-world scheduling problems with hard and soft constraints, particularly those described in [2], we aim to improve Wsat(oip). We introduce three techniques to the existing algorithm. The first is a method of making biased moves in attempting to fix possible discrepancies between the current variable assignment and an optimal solution, so as to drive the search to the regions in search space where high-quality and optimal solutions locate. These biased moves are devised based on our previous work of backbone guided local search for (maximum) Boolean satisfiability

ity [23]. The second method is a sampling-based aspiration search in order to restrict the search to finding progressively improving solutions, so as to reduce search complexity and increase anytime performance of the resulting algorithm. Our experimental analysis show that this method is particularly effective on problems with hard and soft constraints. The third method is an extension of Hoos’s dynamic noise strategy for Walksat [8] to Wsat(oip), so that the critical parameter of noise ratio of Wsat(oip) does not have to be tuned for each individual problem instance. The resulting enhanced Wsat(oip) algorithm becomes more robust, general and flexible for different applications.

The paper is organized as follows. We first describe in Section 2 our motivating scheduling problem and consider its complexity. We then discuss pseudo Boolean encoding and over-constrained integer programs in Section 3, and briefly describe Walksat and Wsat(oip) in Section 4. We then present the three improving techniques for Wsat(oip) in Section 5. We experimentally evaluate the extended Wsat(oip) algorithm in Section 6, using our scheduling problems and the instances of two scheduling problems from CSPLIB [4]. Finally, we conclude in Section 7.

## 2 Scheduling and Resource Allocation

The specific, motivating scheduling problem of this research is to schedule a large number of training activities for a crew over a period of time, ranging from a few days to a few weeks or months [2]. In such a problem, a trainee needs to finish a set of required activities that requires many trainers and various equipment. These activities are associated with one another by precedent relationships, i.e., one training activity cannot be scheduled until a trainee has finished certain prerequisites. A used equipment (resources) can be reused after some maintenance, which itself is an activity to be scheduled. In addition, individual activities have different importance and carry different penalties if not scheduled. The objective is to schedule as many activities as possible for all the trainees within a gross period of time using the available trainers and equipment so that the penalty of unscheduled activities is minimized. Even though this scheduling problem is not overarchingly sophisticated, it can indeed be viewed as a representative of general scheduling problems with various constraints and being required to optimize an objective function.

At the center of our training scheduling problem, as well as many other similar problems, is a resource allocation problem, i.e., a problem of assigning resources (e.g., trainers and equipment in our scheduling problem) to needy activities. The properties of such an underlying resource allocation problem can help characterize the scheduling problem. The complexity of the former will dominate the complexity of the latter. If the resource allocation problem is difficult, the scheduling problem is doomed to be hard as well.

We now consider a simple, static resource allocation problem that was abstracted from our training scheduling problem at a particular time. We are given a set of  $n$  tasks,  $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$ , and a set of  $r$  resources,  $\mathcal{R} = \{R_1, R_2, \dots, R_r\}$ . Each task requires a certain number of resources in order to execute, which we call resource requirements. Each resource can only be allocate to one re-

		$R_1$	$R_2$	$R_3$
$T_1$	$Q_{1,1}$	0	1	1
	$Q_{1,2}$	1	0	0
$T_2$	$Q_{2,1}$	1	1	0
	$Q_{2,2}$	1	1	0

Table 1: A simple resource allocation problem.

source requirement, and a resource requirement can be met by having one desirable resource allocated to it. We denote the  $q_i$  resource requirements of task  $T_i$  by  $\mathcal{Q}_i = \{Q_{i,1}, Q_{i,2}, \dots, Q_{i,q_i}\}$ . Table 1 shows a small example of resource requirements of two tasks over three resources. An entity of 1 (0) in the table means that a resource can (cannot) be allocated to the corresponding requirement. In general, the available resources may not be sufficient to fulfill every task; and a task carries a penalty, called *task penalty*, if not scheduled. The resource allocation problem is to allocate the resources to the tasks so that the overall penalty of unfulfilled tasks is minimized, which constitutes an optimization problem. If all tasks have equal penalties, it is equivalent to fulfill the maximal number of tasks.

Compared to some other resource allocation problems, for instances the permutation problems considered in [18; 21], our problem has a unique, small structure embedded within a task. A task can be scheduled if and only if all its resource requirements are met. We call this feature *bundled resource requirement*. Furthermore, a pair of resource requirements have an exclusive resource contention in that a resource acquired by one requirement cannot be allocated to the others. We call this feature *exclusive resource contention*. To be convenient, we call the problem *bundled, exclusive resource allocation problem*, or *BERAP* for short.

We now show that BERAP is NP-hard [3]. To this end, we prove that a decision version of the problem is NP-complete [3]. A simple, special decision version of BERAP is the following. Given a set of tasks, each of which has a set of resource requirements, decide if at least  $k$  tasks can be fulfilled. Here we simply consider every task having a penalty one if it is not fulfilled.

**Theorem 1** *BERAP with more than two resource requirements per task is NP-complete.*

*Proof:* To show the NP-completeness of the above decision version of BERAP, we reduce a NP-complete set packing problem [3] to it. Given a collection  $S$  of finite sets and a positive integer  $K \leq |S|$ , set packing is to decide if  $S$  contains at least  $K$  mutually disjoint subsets. Formally, it is to decide if there exists  $S' \subseteq S$  such that  $|S'| \geq K$  and for all  $S_1 \in S'$  and  $S_2 \in S'$ ,  $S_1 \cap S_2 = \emptyset$ . The problem is NP-complete when every subset  $S_i \in S$  has more than two elements. We now reduce an NP-complete set packing problem to our decision BERAP. We map all the elements in the subsets of a set packing problem instance to the resources of BERAP, each subset of the set packing instance to a task of BERAP, and an element in the subset to a resource requirement of the respective task. In other words, the total number of tasks is the number of subsets  $|S|$ , the number of resources is the number of distinct elements in all subsets of  $S$ , and the number of resource requirements of

a task is the number of elements in the corresponding subset. Given  $K \leq |S|$ , the constructed BERAP is to decide if at least  $K$  tasks can be fulfilled. Clearly, a solution to the BERAP is also a solution to the original set packing problem.  $\square$

This NP-completeness result leads to the conclusion that our crew scheduling problem is intractable in the worst case.

### 3 PB Encoding and Integer Programs

A clause of Boolean variables can be formulated as a linear pseudo Boolean (PB) constraint [5; 20], which we illustrate by an example. We start by viewing Boolean value True ( $T$ ) as integer 1, and value False ( $F$ ) as 0. We then map a Boolean variable  $v$  to an integer variable  $x$  that takes value 1 or 0, and map  $\bar{v}$  to  $1 - x$ . Therefore, when  $v = T$ , we have  $x = 1$  and  $1 - x = 0$  which corresponds to  $\bar{v} = F$ . With this mapping, we can formulate a clause in a linear inequality. For example,  $(v_1 \vee \bar{v}_2 \vee v_3)$  can be mapped to  $x_1 + (1 - x_2) + x_3 \geq 1$ . Here, the inequality means that the clause must be satisfied in order for the left side of the inequality to have a value no less than one. In general, the class of linear PB constraints is defined as  $\sum_i c_i \cdot L_i \sim d$ , where  $c_i$  and  $d$  are rational numbers,  $\sim$  belongs to  $\{=, \leq, <, \geq, >\}$ , and the  $L_i$  are literals.

However, a clause in an overconstrained problem may not be satisfied so that its corresponding inequality may be violated. To represent this possibility, we introduce an auxiliary integer variable  $w$  to the left side of a mapped inequality. Variable  $w = 1$  if the corresponding clause is unsatisfied, making the inequality valid;  $w = 0$  otherwise. Since the objective is to minimize the number of violated clauses, it is then to minimize the number of auxiliary variables that are forced to take value 1. To be concrete,  $(v_1 \vee \bar{v}_2 \vee v_3), (v_2 \vee \bar{v}_4)$  can be written as an overconstrained PB formula of minimizing  $W = C_1 \cdot w_1 + C_2 \cdot w_2$ , subject to

$$\begin{cases} x_1 + (1 - x_2) + x_3 + w_1 \geq 1 \\ x_2 + (1 - x_4) + w_2 \geq 1 \end{cases}$$

where  $C_1$  and  $C_2$  are the penalties of the first and second clauses, respectively.

More complex constraint problems, where variables takes integers rather than Boolean values, can be formulated as overconstrained integer programs (OIPs) [20], which are integer linear programs (ILPs) [6] in the sense that they both use inequalities to define the feasible regions of a solution space and aim to optimize an objective function. OIPs differ from ILPs in that OIPs introduce additional, competing soft constraints to encode the overall optimization objective.

A constraint in OIP defines a feasible region for all assignments of the (integer) variables involved. For an assignment that violating a constraint, we can define the distance of the assignment to the boundary of the feasible region specified by the constraint. Such a distance can be measured by the Manhattan distance, the minimum integer distance in the grid space defined by the variable domains. This Manhattan distance was called *score* of the constraint under the given assignment [20]. Obviously, if an assignment satisfies a constraint, then its distance to the constraint boundary is zero.

### 4 The Walksat and Wsat(oip) Algorithms

Wsat(oip) belongs to the family of Walksat-based local search algorithms, each of which follows the same basic procedure

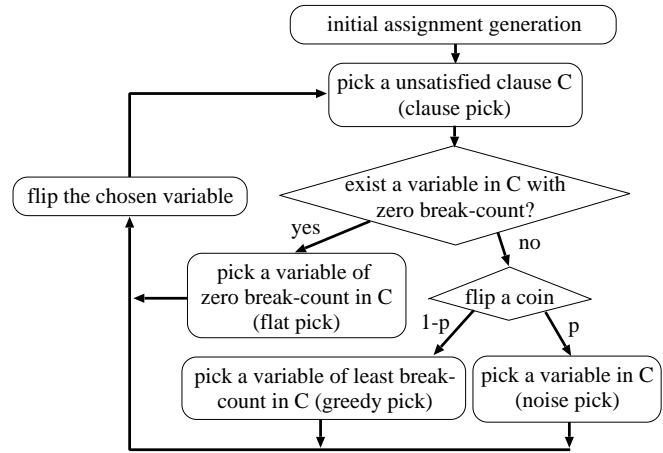


Figure 1: Main operations in a try of WalkSAT.

of the Walksat algorithm [13; 17]. For pedagogic reason, we discuss Walksat first.

#### 4.1 Walksat

The Walksat algorithm starts with an initial random variable assignment and makes moves by changing the value of one selected variable at a time, until it finds a satisfying assignment or after it has executed a predefined maximal number of flips. In the latter case, the best solution encountered so far will be taken as the outcome. Each such unsuccessful attempt is called a *try* or *restart*. The procedure repeats until a maximal number of tries has been attempted.

To select a variable to flip in each step, the effect of changing the current value of a variable is assessed. Changing a variable's value may make some currently satisfied constraint unsatisfied. The numbers of constraints that will be made unsatisfied by changing a variable value is called the *break-count* of the variable at the current assignment. Walksat attempts to change a variable with zero break-count, aiming at not to make the next assignment worse than the current one. To find such a variable with zero break-count, Walksat first selects an unsatisfied clause  $C$ , *uniformly randomly*, from all unsatisfied clauses. This is called **clause pick**. If  $C$  has a variable of zero break-count, Walksat then picks such a variable, *uniformly randomly*, from the ones that qualify (called **flat pick**). If no zero break-count variable exists in  $C$ , Walksat then makes a random choice. With probability  $p$  it chooses, *uniformly randomly*, a variable from all the variables involved in  $C$  (called **noise pick**); or with probability  $1 - p$  it selects a variable with the least break-count, breaking a tie *arbitrarily* if multiple choices exist (called **greedy pick**). The overall procedure for one try of the algorithm is shown in Figure 1. The algorithm takes three parameters to run, the number of tries, the maximal number of flips in each try, and a probability for noise pick, which is commonly referred to as the *noise ratio* of the algorithm.

#### 4.2 Wsat(oip)

Wsat(oip) was built to solve OIPs by extending the Walksat algorithm to support integer variables and generic constraints such as inequalities. The main extensions and modifications made by Wsat(oip) are the following.

- *Distinguishing hard and soft constraints*: When Choosing a violated constraint  $C$ , Wsat(oip) selects a violated hard constraint with probability  $p_h$  and a violated soft constraint with probability  $1 - p_h$ .
- *Restricted neighborhood*: When choosing a variable whose value to be changed from all the variables associated with the selected constraint  $C$ , the (integer) values that differs from the current value by at most  $d$  will be considered.
- *Tabu search*: When multiple variable-value pairs exist in the greedy choice which make the same amount of improvement to the objective value, break ties first in favor of the one that has been used the least frequently, and then in favor of the one that has not been used the longest.

Note that similar to Walksat, Wsat(oip) still uses a noise ratio, which has to be tuned for every problem instance.

## 5 Improvement and Extensions to Wsat(oip)

Wsat(oip) is not very efficient on large problems. We introduce three extensions to improve its performance.

### 5.1 Backbone-guided biased moves

One observation on local search for SAT problems is that there exist a large amount of plateau regions in the search space where neighboring states all have the same quality. This observation inspired the development of Walksat that “walks” on the plateau, thus the name of Walksat, by making random moves in order to navigate through plateau regions and to hopefully find downfall edges. Such random, sometimes aimless, plateau moves are not very effective. Even though the use of a tabu list can help prevent to visit the recently visited states [7], the algorithm may still have to explore a large portion of a plateau area.

The inefficacy of Walksat’s random moves is exacerbated in Wsat(oip) where non-Boolean variables can have large domains, which lead to larger neighborhoods and thus larger plateau regions. Therefore, it is important to shorten or avoid, if possible, such random moves.

#### The main ideas

Our main idea to address the inefficacy caused by uniformly random moves in Wsat(oip) is to exploit an extended concept of backbone. The backbone variables of an optimization problem are the ones that have fixed values among all optimal solutions; and these backbone variables are collectively called the backbone of the problem. The size of the backbone, the fraction of backbone variables among all variables, is a measure of the constrainedness of a given problem. The concept of backbone variables can be extended to backbone frequencies. The backbone frequency of a variable-value pair is the frequency that the pair appears in all optimal solutions; and the backbone frequency of a variable is the maximum backbone frequency of its values. Specifically, let  $x$  be a variable with domain  $D = \{v_1, v_2, \dots, v_k\}$ , and  $p(x(v_i))$  be the backbone frequency of  $x$  taking  $v_i$ , then the backbone frequency of  $x$  is  $p(x) = \max_{v_i \in D} \{p(x(v_i))\}$ . Thus, a backbone variable must have backbone frequency of one. The backbone frequency of a

variable captures the tightness of the constraints that the variable is involved; the higher the frequency, the more constrained the variable is.

We can apply backbone frequencies to modify random moves in Wsat(oip). If, somehow, we knew the backbone frequencies of the variable-value pairs of a problem, we could construct a “smart” search algorithm by using the backbone frequency information as an oracle to guide each step of Wsat(oip). At each step of the algorithm, we could use the backbone frequencies to change the way in which a variable is chosen to focus on fixing the critically constrained variables that are not currently set correctly.

Unfortunately, obtaining the exact backbone frequencies of a problem requires to find all optimal solutions, thus is more difficult than finding just one solution. To address this problem, the second key idea of backbone guided local search is to estimate backbone frequencies using local minima from a local search algorithm. We simply treat local minima as if they were optimal solutions and compute *pseudo backbone frequencies*, which are estimates to real backbone frequencies. More precisely, we define the pseudo backbone frequency of a variable-value pair as the frequency that the pair appears in all local minima.

The quality of pseudo backbone frequencies depends on the effectiveness of the local search algorithm used. High-quality local minima can be obtained by effective local search algorithms. Even though Wsat(oip) may land on suboptimal solutions with fairly high probabilities, most of the local minima from Wsat(oip) are expect to have large portions of variables set to correct values, so that they contain partial optimal solutions or partial backbone. In this research, we directly adopt Wsat(oip) to collect local minima, and then in return apply the backbone guided search method to the algorithm to improve its performance.

#### Biased moves in Wsat(oip)

Pseudo backbone frequencies can be incorporated in Wsat(oip) to make “biased” moves. Consider an example of two variables,  $x_1$  and  $x_2$ , that appear in a violated constraint and have the same effect under the current assignment, i.e., changing the value of one of them makes the violated constraint satisfied, and both variables have the same break-count or will cause the same number of satisfied constraints unsatisfied if changed. Let  $B$  be the set of backbone variables along with their fixed values in the backbone,  $\mathcal{T}$  be the set of local minima from which pseudo backbone frequencies were computed, and  $v_1$  and  $v_2$  be the current values of  $x_1$  and  $x_2$ . We will prefer to change  $x_1$  over  $x_2$  if under the current assignment,  $P\{(x_1 = v_1) \in B | \mathcal{T}\} < P\{(x_2 = v_2) \in B | \mathcal{T}\}$ , which means that under the current assignment,  $x_1$  is less likely to be part of backbone than  $x_2$ , given the set of local minima  $\mathcal{T}$ . Note that  $P\{(x = v) \in B | \mathcal{T}\}$  is the pseudo backbone frequency of  $x = v$  under the evidence of the set of local minima  $\mathcal{T}$ .

How can the pseudo backbone frequencies be used to alter the way that Wsat(oip) chooses variables? As discussed in Section 4, Wsat(oip) *uniformly randomly* chooses a variable when multiple choices exist. For example, when there are multiple variables with zero break-count, Wsat(oip) chooses one arbitrarily. In backbone guided search, we apply pseudo backbone information to force Wsat(oip) to make random yet

*biased* choices. If two variables can make a constraint satisfied, the variable having a higher backbone frequency will be chosen. In other words, we modify Wsat(oip)’s random strategies in such a way that a backbone or critically constrained variable will be chosen more often than a less restricted variable. To this end, we use pseudo backbone frequencies to help make random biased selections.

Specifically, we apply pseudo backbone frequencies to modify the random choices made in Wsat(oip). The first random choice in Wsat(oip) is constraint pick, where a violated constraint is selected if multiple ones exist. We want to pick, with high probabilities, those variables that are part of the backbone or highly constrained in all optimal solutions. Therefore, we choose a constraint with the largest number of critically constrained variables. We use the pseudo backbone frequencies of variables in an unsatisfied constraint, normalized among the violated constraints involved, to measure the degree of constrainedness of the constraint. We then select an unsatisfied constraint among all violated ones based on their degrees of constrainedness. Specifically, let  $\mathcal{C}$  be the set of unsatisfied constraints, and  $q_c$  the sum of pseudo backbone frequencies of all the variables in a constraint  $C \in \mathcal{C}$ . We then select constraint  $C$ , with probability  $p_c = q_c/Q$ , from all unsatisfied constraints in  $\mathcal{C}$ , where  $Q = \sum_{C \in \mathcal{C}} q_c$  is a normalization factor.

Wsat(oip) uses three other random rules to arbitrarily select a variable after an unsatisfied constraint is chosen (see Section 4 and Figure 1). The flat pick rule chooses a variable from a set of zero break-count variables, if any; the noise pick rule selects one from all variables involved in the chosen constraint; and the greedy pick rule takes a variable among the ones of least break-count. In essence, these rules use the same operation, i.e., picking a variable equally likely from a set of variables. Therefore, we can modify these rules all in the same way by using pseudo backbone frequencies. Let  $\{x_1, x_2, \dots, x_w\}$  be a set of  $w$  variables from which one must be chosen,  $\{v_1, v_2, \dots, v_w\}$  be their best satisfying assignments (the ones satisfying the constraint and having the highest pseudo backbone frequencies), and  $\{p_1, p_2, \dots, p_w\}$  be the pseudo backbone frequencies of variable-value pairs  $\{(x_1 = v_1), (x_2 = v_2), \dots, (x_w = v_w)\}$ . Then we choose  $x_i$  with probability  $p_i / \sum_{j=1}^w p_j$ .

Furthermore, the idea of pseudo backbone frequencies can also be applied to generate an initial assignment for a local search. Specifically, a variable is assigned a particular value with a probability proportional to the pseudo backbone frequency of the variable-value pair.

### The backbone guided Wsat(oip) algorithm

The backbone guided Wsat(oip) algorithm has two phases. The first is the *estimation* phase that collects local minima by running Wsat(oip) with a fixed number of tries. The local minima thus collected are compiled to compute the pseudo backbone frequencies of all variable-value pairs.

The second phase carries out the actual *backbone guided* search, which uses pseudo backbone frequencies to modify the way that Wsat(oip) chooses variables to change. This phase also runs many tries, each of which produces a (new) local minimum. The newly discovered local minima are subsequently added to the pool of all local minima found so far to update the

pseudo backbone frequencies.

## 5.2 Aspiration search

Solving an OIP requires to optimize two (conflicting) objectives, satisfying the maximum number of hard constraints and minimizing a penalty function of soft constraints violated. Two obvious methods can be adopted to make a balance between these two objectives. One is to directly search for a solution by considering hard and soft constraint together, which was suggested and taken in Wsat(oip) [20]. This method attempts to select a variable involved in a hard or soft constraint with probability  $p_h$  or probability  $1 - p_h$ , respectively. Note that the performance of Wsat(oip) depends to a large degree on this parameter. The other method is to satisfy the maximum number of hard constraints first and then try to minimize the total penalty of violated soft constraints. However, based on our experimental experience on Wsat(oip), these two methods do not work very well on large, complex OIPs.

In many real-world constraint problems, our training scheduling problems discussed in Section 2 in particular, the number of hard constraints may be large; finding the best assignment to the variables involved in hard constraints itself may be a costly task. Even if such an optimal solution can be found, it may be too hard to be further extended to an overall assignment of minimal penalty. Therefore, many optimal assignments to the variables in hard constraints must be examined, making the overall search prohibitively costly.

To make Wsat(oip) efficient on OIPs, we propose what we call *aspiration search* strategy, which is controlled by aspiration levels. An aspiration level corresponds to a targeted penalty score; the higher an aspiration level, the lower the targeted penalty score. Given an aspiration level, we first search for an assignment so that the total penalty of unsatisfied soft constraints is no more than the targeted penalty value. When such an assignment is found, we attempt to extend the current partial assignment to satisfy the maximum number of hard constraints. This process of extending a partial assignment to a complete assignment may be repeated many times; and the maximum number  $S_h$  of hard constraints satisfied is recorded. Each such process corresponds to a probing in the search space under the current aspiration level. We then increase the aspiration level and repeat the processes of probing with the objective of finding an assignment that violates no more less than  $S_h$  hard constraints and whose penalty meets the restriction of the current new aspiration level.  $S_h$  is also updated if a better assignment, one violating less hard constraints, is found under the current aspiration level. This means that the overall processes attempt to find progressively better solutions for both hard and soft constraints. If we fail to find an assignment satisfying at least  $S_h$  hard constraints and keeping the penalty above the current aspiration level after a certain number of tries, the algorithm terminates and the best solution found so far is returned.

Aspiration search has several advantages. First, it decomposes an OIP into several decision problems, each of which has a different degree of constrainedness represented by an aspiration level. At a given aspiration level, this strategy also integrates a sampling method, which first probes the search space to reach a partial assignment such that the penalty function is above the aspiration level, with a search for satisfaction of the

hard constraints. Thanks to the partial assignment, the hard constraints can be simplified, as the variables involved in soft constraints are instantiated, so that optimizing hard constraints becomes relatively easier. As a result, aspiration search is able to reduce search cost. Second, the probability  $p_h$  of choosing a variable involved in hard constraints, an important parameter determining the performance of Wsat(oip), disappears, making the algorithm less problem dependent. Third, the aspiration search strategy can interact closely with the backbone-guided local search method, making the latter more effective. Finally, since aspiration search is able to reach progressively better solutions, the suboptimal solutions at various aspiration levels can thus be used as local minima to compute pseudo backbone frequencies, which expedites the process of gathering backbone frequency information.

When applying sampling method, it is desirable, albeit difficult, to know if the current partial assignment at a certain aspiration level can be extended to satisfy at least  $S_h$  hard constraints. Our approach to this problem is to monitor the progress of extending the partial assignment to a full assignment. If the number of violated hard constraints decreases after a fixed number of moves  $M$ , we consider the current partial assignment extensible. Otherwise, the current partial assignment will be abandoned, and another partial assignment above the current aspiration level will be sampled. Note that the performance of the overall search is affected by the fixed number of moves  $M$  within which a better complete assignment must be found. If this number is too large, we may waste too much time on an unsatisfiable deadend; whereas if it is too small, we may miss a satisfiable sample. We develop a dynamic method to adjust this parameter  $M$  in our extended Wsat(oip) algorithm. The detail of this method will be discussed in the next section where we collectively deal with the issues of how to dynamically adjust the parameters of the Wsat(oip) and extended Wsat(oip) algorithm.

### 5.3 Dynamic, adaptive parameters

One limitation of the WalkSAT family of algorithms, including Wsat(oip), is its dependence on a manually set noise ratio, which is the probability of how often a nongreedy move should be taken (see Section 4). In addition, whenever a noise ratio is chosen it will be used throughout the search. It is evident that big progresses can be more easily made at an early stage of a local search than at a late stage. Therefore the noise ratio should be adjusted dynamically depending on where the current search is in the overall search space.

The dynamic noise strategy proposed in [8] for Walksat is one such method. The idea of this strategy is simple: start a local search with the noise ratio equal to zero, and examine the number of violations in the current state every  $\theta m$  flips, where  $m$  is the number of constraints of a given problem, and  $\theta$  a constant. If the number of violations has not decreased since the last time we checked ( $\theta m$  flips ago), the search is assumed to have stagnated, and the noise ratio is increased to  $wp + (1 - wp)\phi$ , where  $wp$  is the current noise ratio and  $\phi$  is another constant. Otherwise, the noise ratio is decreased to  $wp(1 - 2\phi)$ . The discrepancy between the formulas for increasing and decreasing the noise ratio is based on some empirical observations of how Walksat behaves when the noise ratio is too high, compared with how it behaves when the parameter

is too low [8]. We refer to this strategy as Dyna-Walksat for convenience.

Dyna-Walksat uses two parameters,  $\theta$  and  $\phi$ . The difference of using these two new parameters and using the noise ratio in the original algorithm is that these two new parameters do not have to be tuned for every single problem instance; the performance of Dyna-Walksat with the same values for  $\theta$  and  $\phi$  is relatively consistent across different problem instances.

Although Dyna-Walksat was originally designed and tested on Walksat for SAT, we have found it effective as well on Wsat(oip) for pseudo Boolean encoded problems and OIPs. We call Wsat(oip) using the dynamic noise strategy Dyna-Wsat(oip). Following [8] we set  $\theta = 1/6$  and  $\phi = 1/5$  in Dyna-Wsat(oip), which have been found to be effective over a wide range of problem instances. Due to its simplicity and reasonable performance, in the rest of the paper we will use Dyna-Wsat(oip) with  $\theta = 1/6$  and  $\phi = 1/5$  as default parameters in our experimental analysis.

As mentioned at the end of the previous section, another parameter in our extended Wsat(oip) algorithm is the number of moves  $M$  between two consecutive check points for examining progress, if any, made by the algorithm in extending the current partial assignment to a complete one that satisfies the maximum number of hard constraints under the current aspiration level. A good value for parameter  $M$  can be achieved when a good balance is made between the algorithm's ability to find satisfied solutions and its ability to escape from local minima. This leads to our adaptive parameter approach, in which the parameter  $M$  is dynamically adjusted based on progress made or not made, as reflected in the time elapsed since the last improvement made to hard constraints. At the beginning of the search, we give an initial value that is proportional to the number of hard constraints  $m$  to parameter  $M$ . If the number of hard constraints does not decrease over the last  $M$  search steps, we increase  $M$  by  $k_1 m$ , where  $k_1$  is a positive constant less than one. Otherwise, we decrease  $M$  by  $k_2 m$ , where  $k_2$  is another positive constant less than one. In our experiments, we took  $M = \min\{m, K/10\}$  and  $k_1 = k_2 = M/5$ , where  $K$  is the maximum number of moves for a restart.

## 6 Applications and Experimental Evaluation

We implemented an improved and extended Wsat(oip) algorithm that incorporates the backbone-guided biased moves, sampling-based aspiration search and dynamic parameter strategies. We short-handed the improved Wsat(oip) as EWsat(oip). In this section, we report the experimental results, comparing the EWsat(oip) algorithm with its predecessor, the Wsat(oip) algorithm. We carried out our analyses on three different scheduling problems: our crew training scheduling problem (Section 2), progressive party scheduling and basketball tournament scheduling. The last two problems were studied in [20], and also included as benchmark problems in CSPLIB [4], an online repository of CSP problems. All our experiments were run on an AMD Athlon 1900 machine with 2GB memory.

In our experiments, we tried various parameter settings for the Wsat(oip) algorithm, which include the probability  $p_h$  of choosing a hard violated constraint over a soft violated constraint and the size of tabu list. The comparison results below

Problem		Wsat(oip)			EWsat(oip)		
$n$	$m$	unsat	penalty	time	unsat	penalty	time
11520	16308	12.43	10380	287.4	<b>12.08</b>	<b>9330</b>	<b>167.6</b>
21240	34908	10.58	8550	824.7	<b>8.05</b>	<b>6270</b>	<b>490.2</b>
21800	37501	3.45	2520	<b>56.0</b>	<b>3.35</b>	<b>2325</b>	56.3
40718	72071	3.95	2970	<b>26.2</b>	<b>3.88</b>	<b>2880</b>	41.2
41496	66892	11.93	10230	1936.4	<b>8.70</b>	<b>6810</b>	<b>965.2</b>
79580	143896	4.10	3120	514.5	<b>3.97</b>	<b>2963</b>	<b>344.3</b>

Table 2: Comparison on crew training scheduling, where  $n$  and  $m$  are the numbers of variables and clauses, respectively; *unsat* is the average number of violated hard constraints, *penalty* the average penalty score, and *time* the average CPU time in seconds. The better results between the two algorithms are in bold.

are for the best parameters for Wsat(oip). Specifically, on our crew scheduling problems, the best probability  $p_h$  is 99%, and on the party scheduling and basketball scheduling problems, the best  $p_h$  is 90%. To make a fair comparison, we applied dynamic parameter method to automatically adjust the noise ratios for both Wsat(oip) and EWsat(oip), and let EWsat(oip) have the same size of tabu list as used by Wsat(oip), which was set to 4.

### 6.1 Crew training scheduling

The first and main problem we considered is our crew training scheduling problem. The test set consists of six large and many small problem instances, derived from a real application domain involved with a large number of crew members of different specialty and various equipment that requires routine maintenance. These problem instances vary in sizes and degree of constrainedness; the largest instances has 79,580 variables and 143,896 constraints. These problems were collected from overconstrained situations and their hard constraints did not seem to be satisfiable all together. Here we present the results on these six large instances.

In our experiments, we allowed Wsat(oip) and EWsat(oip) to have 200 random restarts for each of their run, and 60,000 maximum moves (variable-value changes) with each restart. The average results comparing these two algorithms over 40 runs on all six problem instances are shown in Table 2. We examined the average minimum number of unsatisfied hard constraints (*unsat*), the average minimum penalties (*penalty*) and the average CPU time (*time*) required to reach solutions of such qualities. As the results show, EWsat(oip) is able to find better solutions with more hard constraints satisfied and lower penalties for all problem instances, and with less execution time on four out of the total six instances.

Additional insights were gained when we examined the anytime performance of the two algorithms on these difficult problems. Figure 2 shows such an anytime comparison on one of the six instances, also averaged over 40 runs. As shown, EWsat(oip) can make significant improvement in an early stage of the search, indicating that it explores more fruitful regions of the search space more effectively than Wsat(oip). The anytime results on the other problem instances were similar to that in Figure 2.

### 6.2 Progressive party scheduling

The Progressive Party Problem (PPP) is to progressively timetable a sequence of parties. There are a total of six criti-

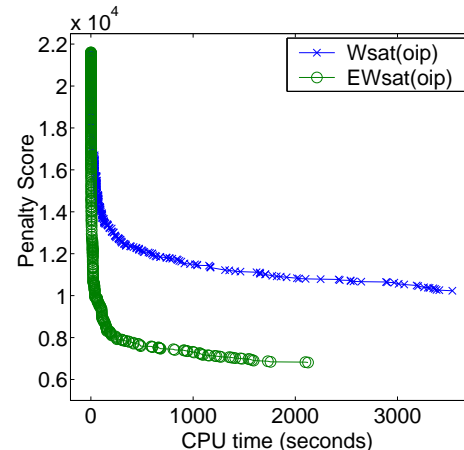


Figure 2: Anytime comparison on a crew scheduling.

cally constrained PPP problem instances in CSPLIB [4], which are all satisfiable. The sizes of these problem instances are relatively small, comparing to the crew scheduling problems, with the number of variables less than five thousand and the number of constraints no more than 32,000. We considered all of them in our experiments. We allowed 100 random restarts in one run of the algorithm, each of which used 60,000 moves. We averaged the results over 40 runs for each of two algorithms. Table 3 shows the median and average CPU times to reach satisfying solutions by Wsat(oip) and EWsat(oip).

On four of the six problem instances, both algorithms take less than one second to finish and have similar performance. On the other two problem instances, EWsat(oip) can reduce median execution time from 15.3 seconds to 9.5 seconds and 44.9 seconds to 34.6 seconds, giving time reductions of 37.9% and 22.9%, respectively.

### 6.3 Basketball tournament scheduling

The ACC Basketball Scheduling Problem (ACC) is to arrange a basketball tournament in the Atlantic Coast Conference. The problem was originally described by Nemhauser and Trick [16]. Walser developed a pseudo Boolean integer linear programming model for these problems [20]. The objective of an encoded ACC scheduling problem is to satisfy all the hard constraints while minimizing the total penalty caused by violated soft constraints. The difficulties of the available

Problem			Wsat(oip)		EWsat(oip)	
name	$n$	$m$	median	average	median	average
ppp:1-12,16	4662	31725	<b>0.175</b>	<b>0.185</b>	0.186	0.190
ppp:1-13	4632	30964	0.406	0.441	<b>0.388</b>	<b>0.423</b>
ppp:1,3-13,19	4608	30348	0.469	0.472	<b>0.388</b>	<b>0.449</b>
Ppp:3-13,25,25	4644	31254	<b>0.625</b>	<b>0.713</b>	0.656	0.718
ppp:1-11,19,21	4602	30179	15.283	15.814	<b>9.453</b>	<b>12.856</b>
ppp:1-9,16-19	4626	30747	44.906	63.553	<b>34.546</b>	<b>58.302</b>

Table 3: Comparison on progressive party scheduling problem, where  $n$  and  $m$  are the numbers of variables and clauses, respectively, and *median* and *average* are the median and average CPU times in seconds. The better results are in bold.

Problem			Wsat(oip)		EWsat(oip)	
name	$n$	$m$	median	average	median	average
acc-tight:2	1620	2520	0.86	<b>1.03</b>	<b>0.77</b>	1.05
acc-tight:3	1620	3249	1.30	2.16	<b>1.26</b>	<b>1.83</b>
acc-tight:4	1620	3285	44.14	61.49	<b>35.35</b>	<b>48.69</b>
acc-tight:5	1339	3052	1171.17	1071.18	<b>603.57</b>	<b>609.05</b>

Table 4: Comparison on ACC basketball scheduling problem, where the legends are the same as in Table 3.

problem instances vary dramatically. Here we only consider four instances of moderate difficulties.

The experiment setup was the same as for the party scheduling problem considered earlier. The median and average times to reach satisfying solutions to these problems are included in Table 4. EWsat(oip) outperformed Wsat(oip) on all these instances except the slightly slower average time on instance acc-tight:2. The performance of EWsat(oip) seems to particularly improve on hard instances. For example, EWsat(oip) reduced the median running time by 48.4% on acc-tight:5, increased from 20.5% on acc-tight:4.

## 7 Conclusions

Wsat(oip) is an extensively applied integer local search algorithm for solving constraint problems with hard and soft constraints which are represented as overconstrained integer linear programs (OIPs). In this paper, we introduced three strategies to improve the performance and applicability of Wsat(oip) in solving complex scheduling problems: biased-move strategy to improve the efficacy of local search by exploiting backbone structures; sampling-based aspiration search to find high quality solutions and improve anytime performance; dynamic parameter adaptation to make Wsat(oip) robust and more applicable to real-world problems. Our experimental results on three large and complex scheduling problems show that our improved Wsat(oip) algorithm significantly improves upon the original Wsat(oip) by finding better solutions on overconstrained problems or finding better or same-quality solutions sooner. We expect that these new methods can be applied to other search algorithms and combinatorial optimization problems.

## Acknowledgment

This research was funded in part by NSF Grants IIS-0196057 and ITR/EIA-0113618, and in part by DARPA Cooperative Agreements F30602-00-2-0531 and F33615-01-C-1897. We

particularly thank Joachim Walser for making the source code of his Wsat(oip) algorithm available to us. Thanks also to USC/ISI Camera group for bringing to our attention the crew scheduling problems and for providing many problem instances studied in the research. We thank Alejandro Bugacov for many helpful discussions.

## References

- [1] H. Dixon and M. L. Ginsberg. Inference methods for a pseudo-Boolean satisfiability solver. In *Proc. of the 18th National Conference on Artificial Intelligence (AAAI-02)*, pages 635–640, 2002.
- [2] M. Frank, A. Bugacov, J. Chen, G. Dakin, P. Szekely, and B. Neches. The marbles manifesto: A definition and comparison of cooperative negotiation schemes for distributed resource allocation. In *Proc. AAAI-01 Fall Symposium on Negotiation Methods for Autonomous Cooperative Systems*, pages 36–45, 2001.
- [3] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, NY, 1979.
- [4] I.P. Gent and T. Walsh. CSPLib: a benchmark library for constraints. Technical report, Technical report APES-09-1999, 1999. Available at <http://csplib.cs.strath.ac.uk/>. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99).
- [5] P. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas*. Springer, 1968.
- [6] F. Hillier and G. Lieberman. *Introduction to Operations Research*. McGraw-Hill, 7th edition, 2002.
- [7] H. H. Hoos. On the run-time behaviour of stochastic local search algorithms for SAT. In *Proc. of the 16th National Conference on Artificial Intelligence (AAAI-99)*, pages 661–666, 1999.

- [8] H. H. Hoos. An adaptive noise mechanism for WalkSAT. In *Proc. the 18th National Conference on Artificial Intelligence (AAAI-02)*, pages 655–660, Edmonton, Canada, July 28-Aug. 1 2002.
- [9] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 1194–1201, 1996.
- [10] H. Kautz and B. Selman. BLACKBOX: A new approach to the application of theorem proving to problem solving. In *Working notes of the Workshop on Planning as Combinatorial Search, held in conjunction with AIPS-98*, 1998.
- [11] H. Kautz and B. Selman. Unifying SAT-based and graph-based planning. In *Proc. the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.
- [12] H. Kautz and J. P. Walser. Integer optimization models of AI planning problems. *Knowledge Engineering Review*, 15:101–117, 2000.
- [13] D. McAllester, B. Selman, and H. Kautz. Evidence for invariants in local search. In *Proc. of the 14th National Conference on Artificial Intelligence (AAAI-97)*, pages 321–326, 1997.
- [14] D. Mitchell, B. Selman, and H. Levesque. Hard and easy distributions of SAT problems. In *Proc. the 10th National Conference on Artificial Intelligence (AAAI-92)*, pages 459–465, 1992.
- [15] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Determining computational complexity from characteristic ‘phase transitions’. *Nature*, 400:133–137, 1999.
- [16] G. Nemhauser and M. Trick. Scheduling a major college basketball conference. *Operations Research*, 46(1), 1998.
- [17] B. Selman, H. Kautz, and B. Cohen. Noise strategies for local search. In *Proc. the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 337–343, 1994.
- [18] B. M. Smith. Modelling a permutation problem. In *Proc. of ECAI-00 Workshop on Modelling and Solving Problems with Constraints*, 2000.
- [19] J. P. Walser. Solving linear pseudo-boolean constraint problems with local search. In *Proc. of the 14th National Conference on Artificial Intelligence (AAAI-97)*, 1997.
- [20] J. P. Walser. *Integer Optimization by Local Search*. Springer, 1999.
- [21] T. Walsh. Permutation problems and channeling constraints. In *Proc. IJCAI-01 Workshop on Modeling and Solving Problems with Constraints*, pages 125–133, 2001.
- [22] W. Zhang. Phase transitions and backbones of 3-SAT and maximum 3-SAT. In *Proc. Intern. Conf. on Principles and Practice of Constraint Programming (CP-01)*, pages 153–167, 2001.
- [23] W. Zhang, A. Rangan, and M. Looks. Backbone guided local search for maximum satisfiability. In *Proc. the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 1179–1184, 2003.