Solving the Employee Timetabling Problem Using Advanced SAT & ILP Techniques

Fadi A. Aloul*, Syed Z. Zahidi, Anas Al-Farra, Basel Al-Roh
American University of Sharjah, Department of Computer Science & Engineering, Sharjah, UAE
*Corresponding author email: faloul@aus.edu

Bashar Al-Rawi
Microsoft, Washington, USA

Abstract—The Employee Timetabling Problem (ETP) is concerned with assigning a number of employees into a given set of shifts over a fixed period of time while meeting the employee’s preferences and organizational work regulations. The problem also attempts to optimize the performance criteria and distribute the shifts equally among the employees. The problem is known to be a complex optimization problem. It has received intensive research during the past few years given its common use in industries and organizations. Several formulations and algorithms based on incomplete search approaches have been proposed to solve employee timetabling problems. In this paper, we propose a complete search approach using Boolean satisfiability (SAT) and integer linear programming (ILP) to solve these problems. The 0-1 ILP model of interest is developed and solved using advanced SAT and ILP solvers. A tool has also been developed to automate the process of producing and solving the ILP model. Experimental results indicate that the proposed approach can effectively handle employee timetabling problems.

Index Terms—Employee Timetabling, Optimization, Scheduling, ILP, Boolean Satisfiability.

I. INTRODUCTION

Employee timetabling problem (ETP) represents an important class of optimization problems in operational research. Given a number of employees and shifts, the goal is to assign employees to shifts while satisfying all of the employees and organizational constraints. The general employee timetabling problem is known to be difficult to solve but has gained wide interest from the research community given its frequent use in practice [1][2]. The problem has also been extended to handle other applications such as educational timetabling [3], crew and transport timetabling[4], sport timetabling [5][6], etc. Many formulations and algorithms have been proposed to solve the employee timetabling problem. Most of these algorithms are based on incomplete search approaches, namely local search techniques[7][8], particle swarm optimization [9][10] [11], genetic algorithms [3], hybrid branch-and-bound [12], heuristic ordering [13], and tabu search [14] [15][16]. Such algorithms cannot prove unsatisfiability or guarantee that a solution is optimal. In other words, if a solution is found, it cannot guarantee that this solution has the best possible optimization cost.

In this paper, we propose a Boolean satisfiability (SAT)-based approach to solve the employee timetabling problem. The approach is complete and hence examines the entire search space defined by the problem to prove that either the problem has no solution, i.e. the problem is unsatisfiable, or that a solution does exist, i.e. the problem is satisfiable. If the problem is satisfiable, the proposed approach will search all possible solutions to find the optimal solution.

Recently, there has been a remarkable growth in the use of Boolean satisfiability (SAT) models and algorithms for solving various complex problems in Engineering and Computer Science. This is mainly due to the fact that SAT algorithms have seen tremendous improvements in the last few years, allowing larger problem instances to be solved in different applications domains. Such applications include FPGA routing [17], global routing [18], verification [19], cryptography [20], power optimization [21], genetics [22], and communications [23]. SAT has also been extended to a variety of applications in Artificial Intelligence including other well-known NP-complete problems such as graph colorability, vertex cover, and Hamiltonian path [24]. Briefly defined, the SAT problem involves a set of Boolean variables and a set of constraints expressed in product-of-sum form, also known as conjunctive normal form (CNF). The goal is to identify an assignment to the variables that would satisfy all constraints or prove that no such assignment exists.

While SAT solvers have traditionally been used to solve decision problems, recently SAT solvers have been extended to handle pseudo-Boolean (PB) constraints [18][25][26][27][28], which are simple inequalities that are equivalent to 0–1 integer linear programming (ILP) constraints. PB constraints are more expressive and can replace an exponential number of CNF constraints. Another key advantage of PB constraints is the ability to express optimization problems which are traditionally handled as ILP problems. Hence, SAT solvers can now handle both decision and optimization problems. Recent studies have shown that SAT solvers can compete with the best available generic ILP solvers in solving 0–1 ILP problems arising in specific applications [18][25].

Even though in recent years we have seen a surge in the application of SAT techniques to assist in finding
solutions to various Engineering problems, we are not aware of the use of SAT-based techniques in solving employee timetabling problems. In this paper, we first show how to model the employee timetabling problem as a SAT, i.e. 0-1 ILP, instance, secondly we develop a tool to automate the process of entering the organizational and employee preferences, producing the ILP model, solving the model and generating the employee schedule, and thirdly we evaluate the performance of the latest SAT and ILP solvers in handling the employee timetabling instances. We show that both SAT and ILP solvers can effectively handle reasonably-sized employee timetabling problems. Furthermore, generic ILP solvers are likely to achieve better results than SAT solvers when handling larger employee timetabling problems.

This paper is organized as follows. Section 2 includes background information on SAT and ILP. Section 3 shows how to formulate the employee timetabling problem as a 0-1 ILP instance. A detailed example is shown in Section 4. Section 5 shows an overview of the developed scheduling tool. Experimental results are presented and discussed in Section 6. The paper is concluded in Section 7.

II. INTRODUCTION

The Boolean satisfiability (SAT) problem involves finding an assignment to a set of binary variables that satisfies a given set of constraints. In general, these constraints are expressed in products-of-sum form, also known as conjunctive normal form (CNF). A CNF formula $\varphi$ on $m$ binary variables $x_1, ..., x_m$ consists of the conjunction (AND) of $m$ clauses $\omega_1, ..., \omega_m$ each of which consists of the disjunction (OR) of $k$ literals. A literal $l$ is an occurrence of a Boolean variable or its complement [29]. Hence, in order to satisfy a formula, each of its clauses must have at least one literal evaluated to true.

Most current SAT solvers [18][25][26][27][28][30][31][32][33] are based on the original Davis-Putnam backtrack search algorithm [34]. The algorithm performs a depth first search process that traverses the space of $2^m$ variable assignments until a satisfying assignment is found, i.e. the formula is satisfiable, or all combinations have been exhausted, i.e. the formula is unsatisfiable. Originally, all variables are unassigned. The algorithm begins by choosing a decision assignment to an unassigned variable. A decision tree is maintained to keep track of variable assignments. After each decision, the algorithm determines the implications of the assignment on other variables. This is obtained by forcing the assignment of the variable representing an unassigned literal in an unresolved clause, whose all other literals are assigned to 0, to satisfy the clause. This is referred to as the unit clause rule. If no conflict is detected, the algorithm makes a new decision on a new unassigned variable. Otherwise, the backtracking process unassigns one or more recently assigned variables and the search continues in another area of the search space. An example of a decision tree is shown in Figure 1.

As an example, a CNF instance $f(a, b, c) = (a + b) \cdot (a + b + c)$ consists of 3 variables, 2 clauses, and 5 literals. The assignment $\{a = 0, b = 1, c = 0\}$ leads to a conflict, whereas the assignment $\{a = 0, b = 0, c = 1\}$ satisfies $f$. Note that a problem with $n$ variables will have $2^n$ possible assignments. An instance with 100 variables will have 1.27e+30 assignments. Assuming a processor that can verify an assignment every 1 nanosecond, the processor will complete testing all assignments in $4e+12$ years.

Despite the SAT problem being NP-Complete [35], several powerful methods have been proposed to expedite the backtrack search algorithm. One of the best methods is known as the conflict analysis procedure [31] and has been implemented in almost all SAT solvers. Whenever a conflict is detected, the procedure identifies the causes of the conflict and augments the clause database with additional clauses, known as conflict-induced clauses, to avoid regenerating the same conflict in future parts of the search process. In essence, the procedure performs a form of learning from the encountered conflicts. Significant speedups have been achieved with the addition of conflict-induced clauses, as they tend to effectively prune the search space. Intelligent decision heuristics and random restarts [30] also played an important role in enhancing the SAT solvers performance. Today, several powerful SAT solvers have been developed and are capable of solving problems consisting of thousands of variables and millions of constraints in a few seconds. Most of these solvers claim competitive results in runtime efficiency and robustness.

Another recent extension to SAT solvers deals with its input format. Restricting the input of SAT solvers to CNF formulas can restrict their usage in various domains. Therefore, researchers have focused on extending SAT solvers to handle stronger input representations. Specifically, SAT solvers [18][25][26][27][28] have been extended to handle pseudo-Boolean (PB) constraints which are linear inequalities with integer coefficients that can be expressed in the normalized form [18] of:

$$f(a, b, c, d) = (a \lor b \lor c) \cdot (a \lor b \lor c) \cdot \overline{(a \lor c \lor d)} \cdot \overline{(a \lor c \lor d)} \cdot (b \lor \overline{c} \lor d) \cdot (b \lor \overline{c} \lor d).$$

![Figure 1. An example of a satisfiable SAT instance showing its corresponding decision tree.](image-url)
where \(a_i, b \in \mathbb{Z}\) and \(x_i\) are literals of Boolean variables. Note that any CNF clause can be viewed as a PB constraint, e.g. clause \((a \lor b \lor c)\) is equivalent to \((a + b + c \geq 1)\).

PB constraints can, in some cases, replace an exponential number of CNF constraints. They have been found to be very efficient in expressing “counting constraints” [18]. Furthermore, PB extends SAT solvers to handle optimization problems as opposed to only decision problems. Subject to a given set of CNF and PB constraints, one can request the minimization (or maximization) of an objective function which consists of a linear combination of the problem’s variables:

\[
\sum_{i=1}^{n} a_i x_i
\]  

This feature has introduced many new applications to the SAT domain. Specifically, all 0-1 integer linear programming (ILP) problems, i.e. ILP problems whose variables are Boolean, can be easily solved now by SAT solvers. Studies have shown that SAT-based 0-1 ILP optimization solvers can in fact compete with some of the best generic-based ILP solvers [18][25].

In this paper, we are interested in using advanced SAT and ILP techniques to solve the employee timetabling problem.

III. PROBLEM FORMULATION

This section develops the 0-1 ILP model for the employee timetabling problem (ETP) and illustrates its applicability using several examples. Consider an organization with \(m\) employees. Assume the organization runs for \(d\) days/week, has \(s\) shifts per day, and needs \(w\) employees to be working in each shift. We first introduce the following decision variables per employee:

\[
E_i = \begin{cases} 
1, & \text{if Employee } i \text{ is working} \\
0, & \text{otherwise}
\end{cases} \quad (3)
\]

\[
D_{ij} = \begin{cases} 
1, & \text{Employee } i \text{ is working on day } j \\
0, & \text{otherwise}
\end{cases} \quad (4)
\]

\[
S_{ijk} = \begin{cases} 
1, & \text{if Employee } i \text{ is working at shift } k \text{ on day } j \\
0, & \text{otherwise}
\end{cases} \quad (5)
\]

Hence the total number of variables will be \(m + md + mds\). Two sets of constraints are added to represent the organizational regulations and employee preferences, respectively. These are described below.

A. Organizational Constraints

The organization needs to make sure that \(w\) employees are assigned to each possible shift. This is expressed in the following constraint:

\[
\sum_{i=1}^{m} S_{ijk} = w \quad \forall j \forall k \quad (6)
\]

The constraint can be customized per shift. For example, the morning shift can have more employees than the afternoon or evening shift.

B. Employee Constraints

The two constraints below define the relationship between the variables \(E_i, D_{ij}\), and \(S_{ijk}\). These constraints are added for all employees. Constraint (7) ensures that if an employee is assigned to any shift, then the employee is working:

\[
(U_{k=1}^s S_{ijk}) \leftrightarrow D_{ij} \quad \forall j \forall i \quad (7)
\]

Constraint (8) ensures that if an employee is working during any day, then the employee is working, i.e. not on vacation, etc.

\[
(U_{j=1}^d D_{ij}) \leftrightarrow E_i \quad \forall i \quad (8)
\]

The following set of constraints deal with the employee preferences. They can be specifically assigned to each employee depending on his/her preferences.

- **Alternating days:** An employee cannot work for two consecutive days.

\[
D_{ij} \rightarrow D_{i(j+1)} \quad \forall i \forall j \neq d \quad (9)
\]

- **Number of shifts per employee:** An employee can work up to \(k_{\text{max}}\) shifts per day. For example, if for security reasons an organization requires employee rotation then the employee does not work all the shifts of a particular day. Note that \(k_{\text{max}}\) can be different for each employee.

\[
(U_{k=1}^s S_{ijk}) \leq k_{\text{max}} \quad \forall j \forall i \quad (10)
\]

- **Number of working days per employee:** An employee must work not less than \(h_{\text{min}}\) days and no more than \(h_{\text{max}}\) days per week.

\[
(S_{ij=1}^d D_{ij}) \geq h_{\text{min}} \quad \forall i \quad (11)
\]

\[
(S_{ij=1}^d D_{ij}) \leq h_{\text{max}} \quad \forall i \quad (12)
\]

- **Off-days:** An employee may be sick or on vacation and unable to work. Assuming employee \(i\) cannot work on day \(j\), this can be expressed using:

\[
D_{ij} = 0 \quad (13)
\]

- **Off-shifts:** An employee may be working part-time or unable of working during specific shifts. Assuming employee \(i\) cannot work in the morning shifts
(indicated by \( k = 1 \)) due to another job. This can be expressed using:

\[
\sum_{i=1}^{d} S_{i,j,k} = 0
\]  

(14)

In order to generate a timetable, the organizational constraints are mandatory, but the employee preferences constraints can be considered optional and can be included when required.

C. Solution Optimization

The constraints described in Section 3.A and 3.B will produce a decision problem that can be solved by all SAT and ILP solvers. However, since advanced SAT and ILP solvers can handle optimization problems, an objective function can be added to further improve the identified schedule and perhaps produce the optimal schedule. The objective function to be optimized can vary depending on the objectives of the organization. For example, the following optimization function can be used to ensure the least number of idle workers:

\[
\text{MAX} \left( \sum_{i=1}^{m} E_i \right)
\]

(15)

On the other hand, the optimization function below will return the schedule using the minimum number of employees and hence the organization’s management can take advantage of that to hire the minimum number of needed employees and reduce labor costs.

\[
\text{MIN} \left( \sum_{i=1}^{m} E_i \right)
\]

(16)

IV. ILLUSTRATIVE EXAMPLE

In this example, we examine a company consisting of two employees. Assume the company is open 3 days a week and includes two shifts per day (e.g. morning and evening shift). Each shift must be assigned to a single employee. As shown in Section 3, the number of variables is \( 2 + 2(3) + 2(3)(2) = 20 \). Figure 2 shows the distribution of variables. We use the variable naming convention used in the previous section. Each node represents a Boolean variable. For example, \( D_{12} \) represents employee 1 working in day 2. \( S_{231} \) represents employee 2 working in the morning shift of day 3.

The company’s organizational constraint indicating that each shift must be assigned to a single employee is expressed as shown in Figure 3.

\[
\begin{align*}
S_{111} + S_{211} & = 1 \\
S_{121} + S_{221} & = 1 \\
S_{131} + S_{231} & = 1 \\
S_{112} + S_{212} & = 1 \\
S_{122} + S_{222} & = 1 \\
S_{132} + S_{232} & = 1 \\
\end{align*}
\]

Figure 3. PB constraints that enforce the organizational regulations.

The constraints in Figure 3 are represented in PB form. Figure 4 shows a set of CNF constraints that enforces the relationship between the \( E \), \( D \), and \( S \) variables.

\[
\begin{align*}
S_{111} & \rightarrow D_{11} \\
S_{112} & \rightarrow D_{12} \\
S_{113} & \rightarrow D_{13} \\
S_{121} & \rightarrow D_{11} \\
S_{122} & \rightarrow D_{12} \\
S_{123} & \rightarrow D_{13} \\
S_{131} & \rightarrow D_{11} \\
S_{132} & \rightarrow D_{12} \\
S_{133} & \rightarrow D_{13} \\
\end{align*}
\]

Logic Expression | CNF Constraint
---|---
\( S_{111} \rightarrow D_{11} \) | \( (S_{111} \lor D_{12}) \land (S_{112} \lor D_{11}) \)
\( S_{112} \rightarrow D_{12} \) | \( (S_{112} \lor D_{12}) \land (S_{113} \lor D_{12}) \)
\( S_{113} \rightarrow D_{13} \) | \( (S_{113} \lor D_{13}) \land (S_{121} \lor D_{13}) \)
\( S_{121} \rightarrow D_{11} \) | \( (S_{121} \lor D_{11}) \land (S_{122} \lor D_{12}) \)
\( S_{122} \rightarrow D_{12} \) | \( (S_{122} \lor D_{12}) \land (S_{123} \lor D_{13}) \)
\( S_{123} \rightarrow D_{13} \) | \( (S_{123} \lor D_{23}) \land (S_{131} \lor D_{23}) \)
\( S_{131} \rightarrow D_{11} \) | \( (S_{131} \lor D_{11}) \land (S_{132} \lor D_{12}) \)
\( S_{132} \rightarrow D_{12} \) | \( (S_{132} \lor D_{12}) \land (S_{133} \lor D_{13}) \)
\( S_{133} \rightarrow D_{13} \) | \( (S_{133} \lor D_{23}) \land (S_{121} \lor D_{23}) \)

Figure 4. CNF constraints that enforce the relationship between the \( E \), \( D \), and \( S \) variables. The CNF constraints (i.e. clauses) are ANDed.

The final set of constraints represents the employees’ preferences. Assume the following working preferences:

- Employee 1 works on alternate days:

\[
\begin{align*}
E_{1} \rightarrow D_{11} \lor D_{12} \\
E_{2} \rightarrow D_{21} \lor D_{22} \\
D_{11} \lor D_{12} \lor D_{13} \rightarrow E_{1} \\
D_{21} \lor D_{22} \lor D_{23} \rightarrow E_{2} \\
\end{align*}
\]

Logic Expression | CNF Constraint
---|---
\( E_{1} \rightarrow D_{11} \lor D_{12} \lor D_{13} \) | \( (E_{1} \lor D_{11}) \land (E_{1} \lor D_{12}) \land (E_{1} \lor D_{13}) \)
\( E_{2} \rightarrow D_{21} \lor D_{22} \lor D_{23} \) | \( (E_{2} \lor D_{21}) \land (E_{2} \lor D_{22}) \land (E_{2} \lor D_{23}) \)
\( D_{11} \lor D_{12} \lor D_{13} \rightarrow E_{1} \) | \( (D_{11} \lor E_{1}) \land (D_{12} \lor E_{1}) \land (D_{13} \lor E_{1}) \)
\( D_{21} \lor D_{22} \lor D_{23} \rightarrow E_{2} \) | \( (D_{21} \lor E_{2}) \land (D_{22} \lor E_{2}) \land (D_{23} \lor E_{2}) \)

Figure 4. CNF constraints that enforce the relationship between the \( E \), \( D \), and \( S \) variables. The CNF constraints (i.e. clauses) are ANDed.

- Employee 1 works on alternate days:

\[
\begin{align*}
D_{11} \rightarrow D_{12} \lor D_{13} \\
D_{12} \rightarrow D_{11} \lor D_{13} \\
\end{align*}
\]

Logic Expression | CNF Constraint
---|---
\( D_{11} \rightarrow D_{12} \lor D_{13} \) | \( (D_{11} \lor D_{12}) \land (D_{12} \lor D_{13}) \)
\( D_{12} \rightarrow D_{11} \lor D_{13} \) | \( (D_{12} \lor D_{11}) \land (D_{12} \lor D_{13}) \)
Employee 2 works for a maximum of 4 shifts per week

\[ PB: S_{211} + S_{212} + S_{221} + S_{222} + S_{231} + S_{232} \leq 4 \]

Employee 1 cannot work during morning shifts

\[ PB: S_{111} + S_{121} + S_{131} = 0 \]

Employee 2 cannot work on day 3

\[ PB: D_{23} = 0 \]

Employee 1 must work at least 2 days per week

\[ PB: D_{11} + D_{12} + D_{13} \geq 2 \]

In summary, the instance consists of 20 variables, 10 PB constraints, and 28 CNF constraints.

V. EMPLOYEE SCHEDULING TOOL

In order to effectively implement the proposed ILP formulation and view the optimal schedules generated by solving the ILP formulation, a tool was developed using Visual Basic. The tool has a user-friendly interface and connects to a selection of the state-of-the-art generic-based ILP and SAT-based 0-1 ILP solvers in the backend.

Through the developed tool, it is possible to set the organizational regulations by configuring the number of employees in the organization, working days and shifts as well as the required number of working employees per shift. In addition, it is possible to set the employee’s preferences such as the off days, off shifts, maximum and minimum number of days to work per week and whether or not the employee works on alternate days.

After reading the user’s input, the tool translates it into a 0-1 ILP instance as described in Section 3. The ILP instance formulation can be generated for a selection of state-of-the-art generic-based and SAT-based 0-1 ILP solvers. The tool currently handles the following 5 solvers: CPLEX [36] and SCIP [37] (generic-based ILP solvers), and BSOLO [27], Pueblo [28] and Minisat+ [26] (SAT-based 0-1 ILP solvers) and can be extended to other solvers. The instance is passed into one of the solvers, as requested by the user. As soon as the solver completes the search, the returned solution is parsed by the tool and, if satisfiable, the satisfying assignment is converted into a visual, tabular schedule. A time-out option for each solver is also set and can be manually adjusted by the user. In the case of a time-out the best identified solution will be displayed in addition to the time-out notification message. Note that some instances can have more than one optimal solution. While the solvers typically return the first identified optimal solution, the solvers can always continue searching and find the remaining optimal solutions. However, this will require additional search time.
VI. EXPERIMENTAL RESULTS

In this section, we evaluate the use of ILP and SAT solvers to solve the employee timetabling problem. As mentioned earlier, five solvers were used to evaluate the effectiveness of the proposed method and developed model. The solvers include the: (1) The SAT-based 0-1 ILP solvers BSOLO [27], Pueblo [28] and Minisat+ [26] and (2) the generic-based ILP solvers CPLEX [36] and SCIP [37]. The experiments were conducted on a Pentium Xeon 3.2 GHz machine, equipped with 4 GBytes of RAM, and running Linux. The runtime limit was set to 1000 seconds for each instance.

Testing was carried out for varying scales of organizational settings, ranging from a small-size setting of only 5 employees, working 3 days a week, 1 shift per day, with 2 employees required to be working each shift up to a large scale organization of up to 50 employees working 7 days a week, 6 shifts per day, with 6 employees required to be working in each shift. For each setting, 5 instances were generated and tested on all selected solvers and the averages of the times taken. In each case, for each employee, unique employee preferences were configured, randomizing the off-days, off-shifts, minimum and maximum number of working days and also whether or not the employee was required to work only on alternate days. The tests are conducted with the objective function given in equation (15), focusing on ensuring that the maximum number of employees are ‘working’. The results are shown in Table 1.

From the results, it can be seen that large instances up to 50 employees working a full-time week, are handled by all solvers, except Minisat+, in less than a second. CPLEX is clearly the fastest solver, from the selected set of solvers, followed by Pueblo and BSOLO with Minisat+ being the slowest of the selected set. The graph in Figure 10 shows how the time taken by each solver to solve the instances, increases with the size of the instance.
TABLE I. Runtine Results for Solving the Employee Timetabling Problem

<table>
<thead>
<tr>
<th>Organizational Setting</th>
<th>Solver Times (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPLEX</td>
</tr>
<tr>
<td>#E #D #S</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.0003</td>
</tr>
<tr>
<td>10</td>
<td>0.0026</td>
</tr>
<tr>
<td>20</td>
<td>0.0084</td>
</tr>
<tr>
<td>30</td>
<td>0.0264</td>
</tr>
<tr>
<td>40</td>
<td>0.0918</td>
</tr>
<tr>
<td>50</td>
<td>0.095333</td>
</tr>
</tbody>
</table>

Figure 10. Comparison of solver search times.

VII. CONCLUSIONS

Recently, Boolean satisfiability (SAT) techniques have been drawing wide research interest and has been shown to be successful in various applications in Engineering and Computer Science. In this paper, we present a complete SAT-based solution to the employee timetabling problem. The proposed approach utilizes advanced generic-based and SAT-based integer linear programming (ILP) solvers to find an optimal schedule that satisfies the organization’s regulations and meets the employees’ preferences. The proposed approach can be used to optimize a given objective function such as minimizing labor costs or maximizing fairness among employees. We show how to formulate the employee timetabling problem as a 0-1 ILP instance. We also develop a tool to automate the process of entering the organization information and retrieving the employee schedule. The approach was tested on organizations with various sizes using several state-of-the-art SAT and ILP solvers. Results indicate SAT and ILP solvers can effectively handle reasonably-sized employee timetabling problems. The proposed approach is complete and is guaranteed to find the optimal solution given enough time and memory resources. It will find the required timetable, or will indicate that no timetable exists that meets the current organization’s conditions.

REFERENCES


