Application of SAT to the Simple Assembly Line Balancing Problem with Power Peak Minimization

Matthieu Py, Arnauld Tuyaba, Laurent Deroussi, Nathalie Grangeon, Sylvie Norre

Application of SAT to the Simple Assembly Line Balancing Problem with Power Peak Minimization

Matthieu Py[1,*], Arnauld Tuyaba[1], Laurent Deroussi[1], Nathalie Grangeon[1] and Sylvie Norre[1]

[1]Université Clermont Auvergne, CNRS, Clermont Auvergne INP, Mines Saint-Etienne, LIMOS, 63000 Clermont-Ferrand, France

Abstract

In this article, we are interested in the SALB3PM problem (Simple Assembly Line Balancing Problem with Power Peak Minimization), a challenging optimization problem in manufacturing environments. We propose to solve this problem by decomposing it as a sequence of (Satisfiability Problem) instances, each addressed by a specialized SAT oracle. Through a series of experiments conducted on state-of-the-art SALB3PM instances, we empirically validate the efficacy and relevance of our proposed methodology.

Keywords

Optimization Problem, Industry 4.0, Assembly Line Balancing, SALBP, SALB3PM, Integer Linear Programming, Constraint Programming, SAT

1. Introduction

Assembly line balancing problems are important industrial issues, notably modeled by the SALBP problems studied in the literature [1]. In particular, Boysen *et al.* [2] state that *one could say that SALBP is the pendant for the production domain what the traveling salesman problem is for the transportation area.* The SALBP problem considers a set of tasks, each task must be handled by one of the workstations for a certain period of time. Within the SALBP problem, we can cite the most important variants:

- SALBP-F: Given the set of tasks, the workstations and the cycle time (the sum of the task times assigned to a workstation must not exceed the cycle time), we want to know if it is possible to assign each task to a workstation in order to respect each constraint (duration of the tasks, precedence constraints, non-overlapping between tasks, ...). It is only a feasibility problem.

- SALBP-1: Given the tasks and the cycle time, we want to choose and minimize the number of workstations such that the SALBP-F problem is feasible with that number of workstations.

- SALBP-2: Given the tasks and the workstations, we want to choose and minimize the cycle time such that the SALBP-F problem is feasible with that cycle time.

- SALBP-E: Given the tasks, two feasible ranges for the number of workstations and the cycle time, we want to minimize the line capacity (the number of workstations multiplied by the cycle time).

Example 1. *We consider the following example with* 4 *tasks,* 3 *workstations and with a cycle time* $c = 5$*. The duration of each task is respectively 5 units of time, 2, 3 and 3. This instance is feasible for the SALB-F problem, as proven in Figure 1.*

15th Pragmatics of SAT international workshop, a workshop of the 27th International Conference on Theory and Applications of Satisfiability Testing, August 20, 2024, Pune, India

*Corresponding author.

matthieu.py@uca.fr (M. Py)

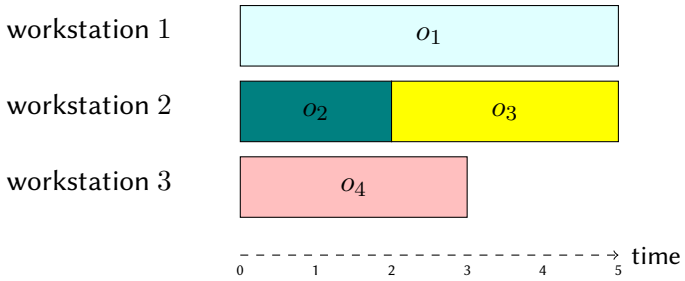

Figure 1: Feasible solution for the SALBP-F problem with 4 tasks, 3 workstations, a cycle time $c = 5$

Due to climate change, new variants of SALBP problems have emerged, in order to take into account the energy impacts of the choices made in the balancing of the assembly line. In this article, we are interested in one particular variant, named the SALB3PM problem, which was proposed in [3]. In this variant, each task requires a constant energy consumption throughout the duration of its execution. The aim is now to minimize the maximum energy consumption per unit of time (called Power Peak) needed to complete all tasks. For this problem, resolution methods using linear programming were used in particular [3, 4]. Here we propose a new approach to solve this problem, based on the progress made in recent decades in solving the Boolean Satisfiability Problem (SAT) [5].

More precisely, we propose an algorithm to solve the SALB3PM problem, based on a sequence of calls to a SAT oracle. Initially, we model the feasibility problem in the form of a propositional formula in conjunctive normal form, without taking into account the energetic elements of the problem, and we call an oracle capable of solving the SAT problem on this instance. Then, we gradually add new constraints to take into account the energy impacts of our decisions and thus obtain better quality solutions from the point of view of energy consumption.

The paper is organized as follows. Section 2 presents the SALB3PM problem. Section 3 describes an existing integer linear program. Section 4 defines and introduce the SAT problem. Section 5 describes our approach to solve the SALB3PM problem using SAT. Section 6 shows the execution of our approach on an illustrative example. Section 7 talks about the computational experiments of our method. Finally, we conclude in Section 8.

2. The SALB3PM Problem

The SALB3PM problem (Simple Assembly Line Balancing Problem with Power Peak Minimization) is an optimization problem which consists, given a set of tasks and a set of workstations, of choosing how to schedule the tasks on the workstations in order to minimize the power peak of the obtained schedule [3, 4, 6, 7].

The data of the SALB3PM problem are the following:

- A set O of n tasks, where:
  - Each task $j \in O$ has a processing time t_j
  - Each task $j \in O$ has a power consumption w_j applied at each unit of time where this task is running

- A set M of workstations ordered from workstation 1 to workstation m

- A cycle time c: the length of the working time to accomplish every task. Time is discretized into c elementary periods numbered from 0 to $(c-1)$, i.e. $T = \{0, ..., c-1\}$. All durations are assumed to be multiples of periods and we will assume that the start of the execution of task always coincides with the start of an elementary period. We note $T^j = \{0, ..., c - t_j\}$ the set of time units where it is possible to start task $j \in O$ in order to end it before the cycle time end.

- A set of precedence constraints P: if a task $i \in O$ precedes a task $j \in O$, which is noted $i \prec j$, consequently:
  - either task i is running on a workstation whose number is strictly smaller than the workstation where task j is running,
  - or tasks i and j are running on the same workstation, but task i is running before task j.

In the SALB3PM problem, we have to choose which workstation takes which task, and to choose the order of running tasks on each workstation, in order to minimize the power peak of the solution.

Example 2. *We consider the following example (similar to example 1 but with additional data) with* 4 *tasks,* 3 *workstations, a cycle time* $c = 5$ *and lexicographic precedences* $o_1 \prec o_2 \prec o_3 \prec o_4$*:*

Table 1
Processing time and power consumption of each task

Task	o_1	o_2	o_3	o_4
Processing time	5	2	3	3
Power consumption	4	4	2	4

Let's see below a feasible solution for the SALB3PM problem on Figure 2 (the power consumption by unit of time of a task is represented by its height). At the beginning, tasks o_1 *(on workstation* 1*),* o_2 *(on workstation* 2*) and* o_4 *(on workstation* 3*) are running at the same time, with a total power consumption of* $4 + 4 + 4 = 12$ *at each time. Then, task* o_2 *ends and task* o_3 *starts (on workstation* 2*), now with a total power consumption of* $4 + 2 + 4 = 10$ *at each time. Finally, tasks* o_4 *ends (on workstation* 3*), now with a total power consumption of* $4 + 4 = 8$ *at each time. The power consumption peak was on the beginning with a power peak of* 12*, so the value of the solution is* 12*. Remark that this schedule is a semi-active one: on each workstation, there is no downtime before the end of the execution on the last task on this workstation.*

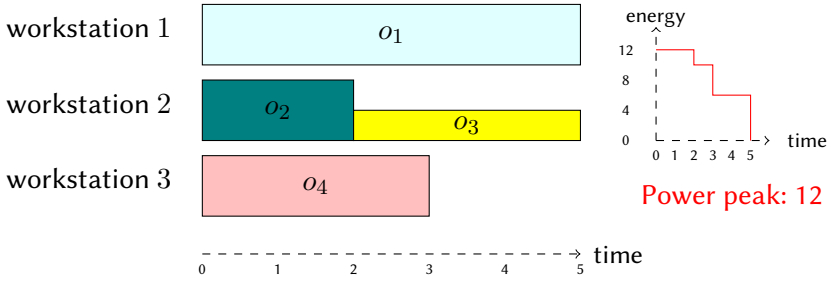

Figure 2: Feasible solution for SALB3PM

Let's see below an optimal solution for the SALB3PM problem. At the beginning, tasks o_1 *(on workstation* 1*) and* o_2 *(on workstation* 2*) are running at the same time, with a total power consumption of* $4 + 4 = 8$ *at each time. Finally, task* o_2 *ends (on workstation* 2*) while task* o_3 *(on workstation* 2*) and task* o_4 *(on workstation* 2*) start, now with a total power consumption of* $4 + 2 + 4 = 10$ *at each time. The power consumption peak was at the end with a power peak of* 10*, so the value of the solution is* 10*. This schedule is no longer a semi-active one, because there is a downtime on workstation 3.*

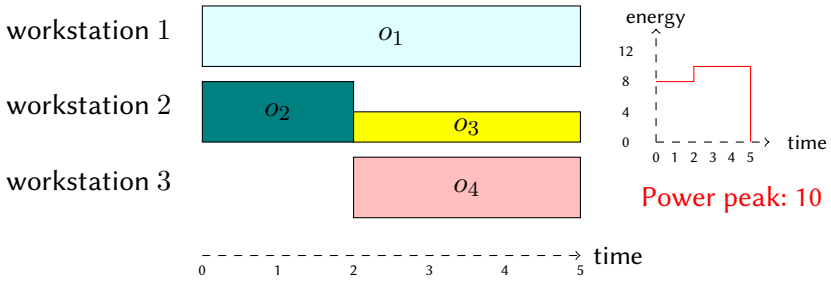

Figure 3: Optimal solution for SALB3PM

3. Existing Integer Linear Program for the SALB3PM Problem

In [3], the SALB3PM problem is modeled using integer linear programming as in Figure 4. The decision variables are the following:

- Assignment variables $X_{j,k}$: Given a task $j \in O$ and a workstation $k \in M$, variable $X_{j,k}$ is equal to 1 if and only if task j is assigned to workstation k.
- Scheduling variables $S_{j,t}$: Given a task $j \in O$ and a time slot $t \in T^j$, variable $S_{j,t}$ is equal to 1 if and only if task j starts at time slot t.
- Power peak variable W_{max}: an upper bound on the power consumption peak.

The first line of the integer linear program is the optimization function while the other lines are constraints:

1. *Objective function* : We have to minimize the power consumption peak.
2. Each task must be assigned to exactly one workstation.
3. The sum of the processing times of every task assigned to the same workstation can't exceed the cycle time.
4. Given a precedence constraint $i \prec j$, task i can't be assigned to a workstation whose number exceeds the number of the workstation on which j is assigned.
5. Each task must start one and only one time.
6. Given a precedence constraint $i \prec j$, if both tasks are assigned to the same workstation, then task i must start before task j.
7. It is not possible to have two different tasks running on the same workstation at the same time (*non-overlap constraint*).
8. For each time slot, the peak energy consumption is greater than the total energy consumption of the tasks scheduled in this time slot.

$$min\ W_{max} \tag{1}$$

$$s.t.\ \sum_{k \in M} X_{j,k} = 1 \qquad \forall j \in O \tag{2}$$

$$\sum_{j \in O} t_j \times X_{j,k} \leq c \qquad \forall k \in M \tag{3}$$

$$X_{j,k} \leq \sum_{h \in M : h \leq k} X_{i,h} \qquad \forall i \prec j \in O, k \in M \tag{4}$$

$$\sum_{t \in T^j} S_{j,t} = 1 \qquad \forall j \in O \tag{5}$$

$$S_{j,t} \leq \sum_{\tau=0}^{t-t_i} S_{i,\tau} + 2 - X_{i,k} - X_{j,k} \qquad \forall i \prec j \in O, k \in M, t \in T^j \tag{6}$$

$$X_{i,k} + X_{j,k} + \sum_{\tau=t-t_i+1}^{t} S_{i,\tau} + \sum_{\tau=t-t_j+1}^{t} S_{j,\tau} \leq 3 \qquad \forall i, j \in O, k \in M, t \in T \tag{7}$$

$$\sum_{j \in O} w_j \times (\sum_{\tau=t-t_j+1}^{t} S_{j,\tau}) \leq W_{max} \qquad \forall t \in T \tag{8}$$

$$X_{i,k},\ \ S_{i,t} \in \{0,1\},\ \ w_{max} \in \mathbb{Z}^+ \qquad \forall i \in O, k \in M, t \in T \tag{9}$$

Figure 4: Integer Linear Program for SALB3PM [3]

4. The SAT Problem

To solve the SALB3PM problem, we will use an approach based on the Satisfiability Problem (SAT). For that purpose, we introduce SAT in this section before presenting our approach in Section 5. The Satisfiability Problem (SAT) checks if a set of constraints can be satisfied or not [8]. The formalism imposed by the SAT problem can be considered more restrictive than the formalism of Integer Linear Programming (ILP). Indeed, both are using decision variables but SAT uses only boolean decision variables. Also, both are using constraints but, in SAT, constraints are represented using boolean expressions instead of linear constraints. Finally, unlike ILP, SAT does not contain any objective function, even it there exists optimization versions of SAT to try to add objective functions such as MaxSAT [9, 10, 11]. Hereafter a formal definition of SAT.

Let X be the set of propositional variables. A literal l is a variable $x \in X$ or its negation $\overline{x}$. A clause c is a disjunction (or set) of literals, i.e., $c = (l_1 \vee l_2 \vee ... \vee l_k)$. A formula in Conjunctive Normal Form (CNF) ϕ is a conjunction (or multiset) of clauses, i.e., $\phi = c_1 \wedge c_2 \wedge ... \wedge c_m$. An assignment $I : X \longrightarrow \{0,1\}$ maps each variable to a Boolean value and can be represented as a set of literals. A literal l is satisfied (resp. falsified) by an assignment I if $l \in I$ (resp. $\overline{l} \in I$). A clause c is satisfied by an assignment I if at least one of its literals is satisfied by I, otherwise it is falsified by I. A CNF formula ϕ is satisfied by an assignment I, that we call model of ϕ, if each clause $c \in \phi$ is satisfied by I, otherwise it is falsified by I. Solving the Satisfiability problem (SAT) consists in determining whether there exists an assignment I that satisfies a given CNF formula ϕ. In the case where such an assignment exists, we say that ϕ is satisfiable, otherwise we say that ϕ is unsatisfiable or inconsistent.

Example 3. *We consider the CNF formula* $\phi = (\overline{x_1} \vee x_3) \wedge (x_1 \vee x_2) \wedge (\overline{x_2} \vee \overline{x_3})$*:*

- *This formula contains 3 variables* x_1*,* x_2 *and* x_3*.*
- *This formula contains 3 clauses (or 3 constraints). First clause claims that variable* x_1 *must be equal to 0 or variable* x_3 *must be equal to 1. Second clause claims that variable* x_1 *must be equal to 1 or*

variable x_2 must be equal to 1. Third clause claims that variable x_2 must be equal to 0 or variable x_3 must be equal to 0.

- *This formula is satisfiable, for instance with assignment $I = \{x_1, \overline{x_2}, x_3\}$, in other words $x_1 = x_3 = 1$ and $x_2 = 0$. Indeed, in this situation, first clause is satisfied thanks to variable x_3, second clause is satisfied thanks to variable x_1 and third clause is satisfied thanks to variable x_2.*

Despite the syntactic restrictions of the propositional formulas, SAT is an unavoidable problem in Computer Science, both in theory and in practice. In theory, SAT is the first problem shown to be NP-complete [12] and is at the center of the complexity theory. In practice, SAT is used to solve many types of problems like model checking [13] or automated theorem proving [14]. This is due to the progress made in the techniques used to solve the SAT problem [5, 15, 16].

5. Solving SALB3PM using SAT

We model the SALB3PM problem in the form of a SAT problem. As SAT is a decision problem but is not an optimization problem, we only model first the real constraints of the problem, i.e. how to find a feasible solution. Figure 5 shows the constraints of the problem under the form of boolean clauses.

Initial SAT model:

$$\bigvee_{k \in M} X_{j,k} \qquad \forall j \in O \quad (1)$$

$$\overline{X_{j,k_1}} \vee \overline{X_{j,k_2}} \qquad \forall j \in O, k_1, k_2 \in M : k_1 < k_2 \quad (2)$$

$$\overline{X_{j,k}} \vee \overline{X_{i,h}} \qquad \forall i, j \in O : i \prec j, k, h \in M : k < h \quad (3)$$

$$\bigvee_{t \in T^j} S_{j,t} \qquad \forall j \in O \quad (4)$$

$$\overline{S_{j,t_1}} \vee \overline{S_{j,t_2}} \qquad \forall j \in O, t_1, t_2 \in T^j : t_1 < t_2 \quad (5)$$

$$\overline{S_{j,t}} \qquad \forall j \in O, t \in T : t \notin T^j \quad (6)$$

$$\overline{X_{i,k}} \vee \overline{X_{j,k}} \vee \overline{A_{i,t}} \vee \overline{A_{j,t}} \qquad \forall i, j \in O : i \neq j, k \in M, t \in T \quad (7)$$

$$\overline{S_{j,t}} \vee A_{j,t+\epsilon} \qquad \forall j \in O, t \in T^j, \epsilon \in [0, t_j - 1] \quad (8)$$

$$\overline{X_{i,k}} \vee \overline{X_{j,k}} \vee \overline{S_{i,t_1}} \vee \overline{S_{j,t_2}} \qquad \forall i, j \in O : i \prec j, k \in M, \; t_1 \in T^i, t_2 \in T^j : t_1 > t_2 \quad (9)$$

Simplification constraints:

$$\overline{X_{j,k}} \qquad \forall j \in O, k \in M : ip(j,k) \quad (10)$$

$$\overline{X_{j,k}} \vee \overline{S_{j,t}} \qquad \forall j \in O, k \in M : p(j,k), t \in T^j : ip(j,k,t) \quad (11)$$

$$A_{j,t} \qquad \forall t \in [c - t_i, t_i - 1] \quad (12)$$

Optimization constraints:

$$\bigvee_{j \in C} \overline{A_{j,t}} \qquad \forall C \in \mathbb{C}, t \in T \quad (13)$$

Figure 5: Representation of SALB3PM as a propositional formula

Decision variables used by SAT are the following ones, where the two first set of variables were also used as ILP variables. Remember that we use only boolean decision variables in SAT:

- Assignment variables $X_{j,k}$: Given a task $j \in O$ and a workstation $k \in M$, variable $X_{j,k}$ is equal to 1 if and only if task j is assigned to workstation k.

- Scheduling variables $S_{j,t}$: Given a task $j \in O$ and a time slot $t \in T^j$, variable $S_{j,t}$ is equal to 1 if and only if task j starts at time slot t.

- Activity variables $A_{j,t}$: Given a task $j \in O$ and a time slot $t \in T$, variable $A_{j,t}$ is equal to 1 if and only if j is currently running in time slot t. These variables will be important to forbid some tasks to be executed at the same time.

The first lines of the propositional formula are the constraints of the SALB3PM problem:

1. Each task must be assigned to at least one workstation.

2. Each task must be assigned to at most one workstation.

3. Given a precedence constraints $i \prec j$, task i can't be assigned to a workstation whose number exceeds the number of the workstation on which j is assigned.

4. Each task must start at least one time.

5. Each task must start at most one time.

6. A task can't start if it can't consequently finish before cycle time.

7. It is not possible to have two different tasks running on the same workstation at the same time. (*non-overlap constraint*)

8. If a task starts its execution, it must then be active during its execution time.

9. Given a precedence constraint $i \prec j$, if both tasks are assigned to the same workstation, then task i can't start after task j.

Notice how the propositional formula is bigger than the integer linear program in Section 3. For example, the first set of constraints (line 2) of the ILP appears $|O|$ times in the model while the SAT-equivalent constraints are constraints (1) and (2) and appear around $|O|^2 + |O|$ times. Therefore, we use a preprocess technique to eliminate trivially infeasible solutions and useless constraints, which is useful for space consideration (and has also small impact on the running time of our approach). This preprocess technique adds constraints 10, 11 and 12 and when we add constraints 1 to 9, we immediately remove it if there is a contradiction with these new constraints. In this preprocess technique, we compute an earliest possible starting date and assigned workstation for each task. Naively each task can start at the first time slot in the first workstation. Then, we explore the precedences constraints and we update these two data, in order to eliminate trivially impossible decisions for each task. We apply the same process computing latest possible starting date and assigned workstation for each task. We get the following two set of constraints, plus a trivial one saying that long tasks are necessarily active in the middle of the time horizon:

10 If a task j can't be assigned to a workstation k because of precedence constraints, we prohibit this decision (we note $ip(j,k) = 1$).

11 If a task j can't be assigned to a workstation k at a time slot t because of precedence constraints, we prohibit this decision (we note $ip(j,k,t) = 1$).

12 If a task duration is more than half of cycle time, this task is necessarily active in the middle of the time horizon.

Solving the SAT problem on the formula (excluding constraint (13)) makes it possible to find a solution which respects all the constraints of the SALB3PM problem. It can be seen as an algorithm able to solve the SALB-F Problem. To determine the scheduling with a minimum energy peak, we restart our algorithm as many times as needed by integrating new constraints into the model. Indeed, each time the SAT oracle returns a feasible solution, we analyze the computed solution to determine its energy peak and what are the tasks executed at the same time causing an energy peak greater or equal than the value of the currently best-known solution. Then, we introduce optimization constraints to forbid these tasks to be executed at the same time. These optimization constraints correspond to line (13) of the mathematical model where $\mathbb{C}$ contains the set of tasks that we no longer want to be active at the same time. Set $\mathbb{C}$ is initially empty and gradually expands through to the analysis of solutions proposed by the SAT oracle. Each time we add new optimization constraints, we call again the SAT oracle with these new optimization constraints. When the SAT oracle returns that the feasible problem is no longer feasible, the best solution computed from the beginning is an optimal solution for the SALB3PM problem.

Our approach is resumed in Algorithm 1 after. At the beginning, we compute the initial SAT model including the initialization of the empty set of optimization constraints $\mathbb{C}$ (*line 1*) and we note that we currently know no solution (*lines 2 and 3*). Then, we call the SAT oracle for the first time, in order to compute a solution to the feasibility problem (*line 4*). If the feasibility problem is not feasible, we end with no solution (*lines 5 to 7*). Otherwise, we start the main loop of our algorithm, running as long as the SAT oracle is able to find a feasible solution (*line 8 to 16*). At each iteration of the loop, we compute the energy peak of the current solution (*line 9*) and we save it if it is the best solution ever met (*lines 10 to 13*). We analyze the current solution to get at least one (possibly more) set of tasks executed at the same time in the current solution and causing a power peak greater or equal than the value of the currently best solution known. For each of these set of tasks, we add a set of clauses to prohibit this set of tasks from being active at the same time, i.e. we add new optimization constraints to the model by the enlargement of set $\mathbb{C}$ (*line 14*). Then, we call again the SAT oracle to get a new feasible solution. If the SAT oracle returns a new feasible solution, we make another round of the loop, otherwise we return the best known solution as an optimal solution (*line 17*).

Algorithm 1 SALB3PM problem solving method using SAT

Require: Instance i of the SALB3PM problem
Ensure: Optimal solution of i with its value

1: $model \leftarrow$ compute_initial_model(i)
2: $bestSolution \leftarrow \emptyset$
3: $bestValue \leftarrow +\infty$
4: $(status, currentSolution) \leftarrow$ solve_SAT_problem$(model)$
5: **if** status = "UNFEASIBLE" **then**
6: **return** ("NO FEASIBLE SOLUTION", bestSolution, bestValue)
7: **end if**
8: **while** status = "FEASIBLE" **do**
9: $currentValue \leftarrow$ compute_value_of_solution$(i, currentSolution)$
10: **if** $currentValue < bestValue$ **then**
11: $bestSolution = currentSolution$
12: $bestValue = currentValue$
13: **end if**
14: $model \leftarrow$ add_new_constraints$(model, i, currentSolution)$
15: $(status, currentSolution) \leftarrow$ solve_SAT_problem$(model)$
16: **end while**
17: **return** ("OPTIMAL SOLUTION", bestSolution, bestValue)

6. Illustrative example

In this section, we show how our approach works on an example. To this end, we consider example 2 with 4 tasks, 3 workstations, a cycle time $c = 5$ and lexicographical precedences $o_1 \prec o_2 \prec o_3 \prec o_4$:

Table 2
Tasks, processing time and power consumption

Task	o_1	o_2	o_3	o_4
Processing time	5	2	3	3
Power consumption	4	4	2	4

We start by the creation of the initial SAT model as described in Section 5. In particular, there is no restriction about tasks that should not be allowed to be active at the same time, i.e. $\mathbb{C} = \emptyset$. Then, we call the SAT solver for the first time in order to find a feasible solution. At the first iteration, the SAT solver returns that there exists the following feasible solution:

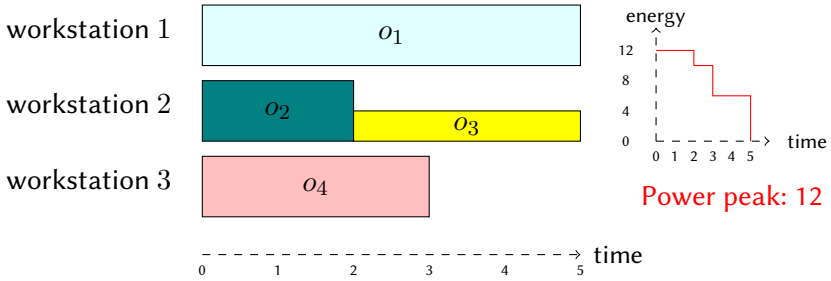

Figure 6: Feasible solution computed by the first SAT oracle call

The proposed solution has a power consumption peak of 12, because tasks o_1, o_2 and o_4 are executed at the same time. We consequently add the following new constraints, in order to prohibit these tasks to be executed at the same time:

$$\overline{A_{o_1,t}} \vee \overline{A_{o_2,t}} \vee \overline{A_{o_4,t}} \qquad \forall t \in T$$

Now, tasks o_1, o_2 and o_4 can no longer be active at the same time, i.e. $\mathbb{C} = \{\{o_1, o_2, o_4\}\}$. We call again the SAT solver for the second time in order to find a feasible solution. At the second iteration, the SAT solver returns that there exists the following feasible solution:

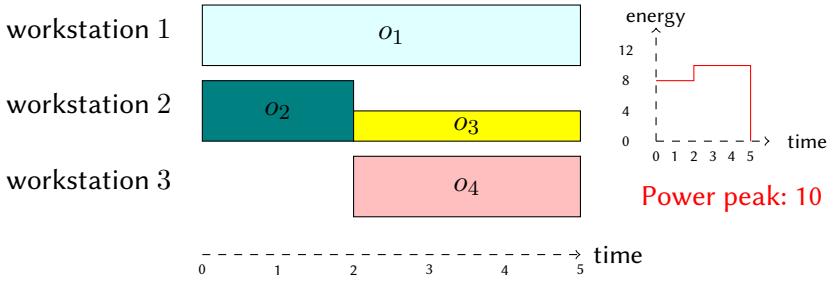

Figure 7: Feasible solution computed by the first SAT oracle call

The proposed solution has a power consumption peak of 10, because tasks o_1, o_3 and o_4 are executed at the same time. We consequently add the following new constraints, in order to prohibit these tasks to be executed at the same time:

$$\overline{A_{o_1,t}} \vee \overline{A_{o_3,t}} \vee \overline{A_{o_4,t}} \qquad \forall t \in T$$

Now, tasks o_1, o_3 and o_4 can no longer be active at the same time, i.e. $\mathbb{C} = \{\{o_1, o_2, o_4\}, \{o_1, o_3, o_4\}\}$. We call again the SAT solver for the third time in order to find a feasible solution. At the third iteration, the SAT solver claims that it is no longer possible to compute a feasible solution.

Consequently, among all the computed solutions, the solution providing the minimum power consumption peak is an optimal solution. In this example, the best solution was proposed at the second iteration with a power consumption peak of 10. Consequently, the optimal power consumption peak is 10 in this example, and the proposed solution is optimal.

7. Experiments

We experimented our algorithm on several instances from the literature, with for each instance the results obtained by implementing the existing model on CPLEX [3] and the results obtained with our algorithm. The experiments are performed on Dell computer with Intel(R) Core(TM) i5-8400T processor (clocked at 1.70 GHz) under Windows 10. The underlying SAT problem is solved using the Sat4j optimization library [17] which includes the SAT Solver Glucose [18]. We give, for each instance, the number of tasks n, workstations m, the cycle time c and, for each method, the energy peak of the best solution computed, the execution time (at most 1 hour) and its status (optimal or not).

Table 3
Comparison of ILP and SAT approaches on some instances

Instance	Data			ILP Results			SAT Results		
	n	m	c	Peak	Time	Status	Peak	Time	Status
mertens-1	7	6	6	**104**	**0.01 s**	Optimal	**104**	0.23 s	Optimal
mertens-2	7	2	18	**49**	**0.09 s**	Optimal	**49**	0.24 s	Optimal
bowman-1	8	5	20	**164**	**0.05 s**	Optimal	**164**	0.23 s	Optimal
jaeschke-1	9	8	6	**248**	**0.02 s**	Optimal	**248**	0.21 s	Optimal
jaeschke-2	9	3	18	**78**	**0.16 s**	Optimal	**78**	0.28 s	Optimal
jackson-1	11	8	7	**179**	**0.08 s**	Optimal	**179**	0.25 s	Optimal
jackson-2	11	2	94	**65**	1.05 s	Optimal	**65**	**0.50 s**	Optimal
mansoor-1	11	4	48	**145**	0.69 s	Optimal	**145**	**0.38 s**	Optimal
mansoor-2	11	2	94	**77**	5.03 s	Optimal	**77**	**0.56 s**	Optimal
mitchell-1	21	8	14	**221**	1.44 s	Optimal	**221**	**0.60 s**	Optimal
mitchell-2	21	3	39	**85**	427.94 s	Optimal	**85**	**9.88 s**	Optimal
roszieg-1	25	10	14	**254**	104.36 s	Optimal	**254**	**7.80 s**	Optimal
roszieg-2	25	4	32	**117**	163.59 s	Optimal	**117**	**5.65 s**	Optimal
heskiaoff-1	28	8	138	$\leq$ 290	$\geq$ 3600 s		**251**	**196.69 s**	Optimal
heskiaoff-2	28	3	342		$\geq$ **3600 s**		$\leq$ **107**	$\geq$ **3600 s**	
buxey-1	29	14	25	$\leq$ **292**	$\geq$ **3600 s**		$\leq$ **292**	$\geq$ **3600 s**	
buxey-2	29	7	47	$\leq$ 350	$\geq$ 3600 s		**172**	**137.46 s**	Optimal
sawyer-1	30	14	25	**395**	157.00 s	Optimal	**395**	**2.04 s**	Optimal
sawyer-2	30	7	47		$\geq$ 3600 s		**214**	**257.92 s**	Optimal
gunther-1	35	14	40	**394**	2297.00 s	Optimal	**394**	**2.13 s**	Optimal
gunther-2	35	9	54		$\geq$ 3600 s		**295**	**6.08 s**	Optimal

We see in the Table 3 that our approach finds the optimal solution on more instances (19 instances out of 21) than with the known ILP approach (15 out of 21). In particular, ILP is better than our approach on trivial instances (because the time to write the ILP model is smaller than the time to write the SAT model). On the other hand, on non-trivial instances, our approach is better than the studied ILP model.

In the Table 4, we detail the results obtained by our approach with, for each instance, the number of tasks n, workstations m, the cycle time c, the number of decision variables $\#Var$, the number of constraints $\#Const$ (in the initial model), the energy peak of the best solution calculated (Peak), the

Table 4
Details about our SAT approach on some instances

Instance	Data			SAT Results						
	n	m	c	#Var	#Cons	Peak	Status	#Sol	#SolBB	Time
mertens-1	7	6	6	126	578	104	Optimal	9	9	0.23 s
mertens-2	7	2	18	266	2844	49	Optimal	3	3	0.24 s
bowman-1	8	5	20	360	2601	164	Optimal	3	3	0.23 s
jaeschke-1	9	8	6	180	586	248	Optimal	2	2	0.21 s
jaeschke-2	9	3	18	351	4049	78	Optimal	19	18	0.28 s
jackson-1	11	8	7	242	1454	179	Optimal	2	2	0.25 s
jackson-2	11	3	21	495	9 798	65	Optimal	33	31	0.50 s
mansoor-1	11	4	48	1100	23 549	145	Optimal	5	5	0.38 s
mansoor-2	11	2	94	2090	70 925	77	Optimal	4	1	0.56 s
mitchell-1	21	8	14	756	8 189	221	Optimal	100	43	0.60 s
mitchell-2	21	3	39	1701	50 041	85	Optimal	121	114	9.88 s
roszieg-1	25	10	14	950	27 090	254	Optimal	883	562	7.80 s
roszieg-2	25	4	32	1700	62 749	117	Optimal	120	116	5.65 s
heskiaoff-1	28	8	138	7952	1 025 654	251	Optimal	866	865	196.69 s
heskiaoff-2	28	3	342	19 236	3 875 781	≤ 107		82	81	≥ 3600 s
buxey-1	29	14	25	1856	76 114	≤ 292		8720	2523	≥ 3600 s
buxey-2	29	7	47	2929	169 279	172	Optimal	53	52	137.46 s
sawyer-1	30	14	25	1920	93 520	395	Optimal	114	114	2.044 s
sawyer-2	30	7	47	3030	204 757	214	Optimal	63	63	257.92 s
gunther-1	35	14	40	3290	260 795	394	Optimal	156	156	2.13 s
gunther-2	35	9	54	4095	372 766	295	Optimal	60	51	6.08 s

status after at most one hour (optimal or not), the number of different solutions computed, the iteration in which the approach found the best one and the execution time (at most 1 hour). Notice that the number of iterations is usually limited, because our approach can eliminate an exponential number of feasible solutions at each iteration. Among the two instances on which our approach fails to prove optimality in one hour, the instance called *heskiaoff-2* contains more than one million constraints and the approach finds less than one hundred solution in one hour while the instance called *buxey-1* has the line capacity (number of workstations multiplied by cycle time) which is a little bit oversized for this instance, allowing too many different patterns of feasible solutions compared to *buxey-2* which is the same instance with a different line capacity.

8. Conclusion

In this article, we examined the SALB3PM problem, which is a variation of the well-known SALBP problems. This particular problem incorporates an energetic component to address challenges prevalent in the industrial sector, a significant consumer of energy and emitter of greenhouse gases [19, 6]. To solve this problem, we proposed an approach that iteratively uses the SAT problem to solve this optimization problem. This approach has been compared with an existing integer linear programming approach, and provides promising results.

The immediate future work is about increasing the performances of our SAT-based approach and experiment a Max-SAT approach. In particular, it may be possible to upgrade the performances of our approach, in term of time and space, thanks to the existing work about the encoding of cardinality constraints [20, 21, 22]. An other direction is about encoding the objective function as a set of soft clauses as in the MaxSAT Problem [23, 24]. Finally, more distant future work are about using other formulation like the CSP problem [25] and trying to hybridize these approaches with each other or with other heuristic approaches.

References

[1] A. Scholl, C. Becker, State-of-the-art exact and heuristic solution procedures for simple assembly line balancing, European Journal of Operational Research 168 (2006) 666–693.

[2] N. Boysen, P. Schulze, A. Scholl, Assembly line balancing: What happened in the last fifteen years?, European Journal of Operational Research 301 (2022) 797–814.

[3] P. Gianessi, X. Delorme, O. Masmoudi, Simple Assembly Line Balancing Problem with Power Peak Minimization, in: IFIP International Conference on Advances in Production Management Systems (APMS), 2019.

[4] P. Gianessi, X. Delorme, A new ILP Model for a Line Balancing Problem with Minimization of Power Peak, in: 31st European Conference on Operational Research (EURO-2021), 2021.

[5] H. van Maaren, J. Franco, SAT Competitions, http://www.satcompetition.org/, 1992.

[6] D. Lamy, X. Delorme, P. Gianessi, Line Balancing and Sequencing for Peak Power Minimization, IFAC-PapersOnLine 53 (2020) 10411–10416. 21st IFAC World Congress.

[7] X. Delorme, P. Gianessi, D. Lamy, A new Decoder for Permutation-based Heuristics to Minimize Power Peak in the Assembly Line Balancing, IFAC-PapersOnLine 56 (2023) 3704–3709. 22nd IFAC World Congress.

[8] A. Biere, M. Heule, H. van Maaren, T. Walsh, Handbook of Satisfiability, volume 336 of *Frontiers in Artificial Intelligence and Applications*, 2 ed., IOS Press, 2021.

[9] J. Davies, F. Bacchus, Solving MAXSAT by Solving a Sequence of Simpler SAT Instances, in: Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming, CP'11, 2011, pp. 225––239.

[10] Z. Fu, S. Malik, On Solving the Partial MAX-SAT Problem, in: Theory and Applications of Satisfiability Testing - SAT 2006, 2006.

[11] M. Py, M. S. Cherif, D. Habet, Proofs and certificates for max-sat, Journal of Artificial Intelligence Research 75 (2022) 1373–1400.

[12] S. A. Cook, The Complexity of Theorem-Proving Procedures, in: Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, 1971, pp. 151–158.

[13] E. M. Clarke, A. Biere, R. Raimi, Y. Zhu, Bounded Model Checking Using Satisfiability Solving, Formal Methods in System Design 19 (2001) 7–34.

[14] B. Konev, A. Lisitsa, A sat attack on the erdos discrepancy conjecture, in: C. Sinz, U. Egly (Eds.), Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Proceedings, volume 8561 of *Lecture Notes in Computer Science*, Springer, 2014, pp. 219–226.

[15] N. Eén, N. Sörensson, An Extensible SAT-solver, in: E. Giunchiglia, A. Tacchella (Eds.), Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers, volume 2919 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 502–518.

[16] A. Biere, K. Fazekas, M. Fleury, M. Heisinger, CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020, in: T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, M. Suda (Eds.), Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions, volume B-2020-1 of *Department of Computer Science Report Series B*, University of Helsinki, 2020, pp. 51–53.

[17] D. L. Berre, A. Parrain, The Sat4j library, release 2.2, Journal on Satisfiability, Boolean Modeling and Computation 7 (2010) 59–64.

[18] G. Audemard, L. Simon, On the Glucose SAT Solver, International Journal on Artificial Intelligence Tools (IJAIT) 27 (2018) 1840001:1–1840001:25.

[19] Y. Wang, L. Li, Time-of-use based electricity demand response for sustainable manufacturing systems, Energy 63 (2013) 233–244.

[20] M. Bofill, J. Coll, P. Nightingale, J. Suy, F. Ulrich-Oltean, M. Villaret, SAT encodings for Pseudo-Boolean constraints together with at-most-one constraints, Artificial Intelligence 302 (2022) 103604.

[21] J. Chen, A New SAT Encoding of the At-Most-One Constraint, in: Proc. of the Tenth Int. Workshop of Constraint Modelling and Reformulation, 2010.

[22] O. Bailleux, Y. Boufkhad, O. Roussel, New Encodings of Pseudo-Boolean Constraints into CNF, in: O. Kullmann (Ed.), Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings, volume 5584 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 181–194.

[23] J. Argelich, C. M. Li, F. Manyà, J. Planes, The first and second max-sat evaluations, Journal of Satisfiability Boolean Modeling and Computation 4 (2008) 251–278. URL: https://maxsat-evaluations.github.io.

[24] C. Li, Z. Xu, J. Coll, F. Manyà, D. Habet, K. He, Combining Clause Learning and Branch and Bound for MaxSAT, in: 27th International Conference on Principles and Practice of Constraint Programming, CP 2021, October 25-29, 2021, volume 210 of *LIPIcs*, 2021, pp. 38:1–38:18.

[25] S. C. Brailsford, C. N. Potts, B. M. Smith, Constraint satisfaction problems: Algorithms and applications, European Journal of Operational Research 119 (1999) 557–581.