Solving Assembly Line Workload Smoothing Problem via Answer Set Programming

Orkunt Sabuncu [*] and Mehmet Cem Şimşek

TED University, Ankara, Turkey
{orkunt.sabuncu,mcem.simsek}@tedu.edu.tr

Abstract. Assembly line balancing problems aim to assign production tasks to workstations or people while satisfying some constraints and optimizing various criteria such as production rate or number of workstations needed. One specific instance is Simple Assembly Line Workload Smoothing Problem (SALBP-WS) that focuses on smoothing workloads of all workstations. This is important for establishing equity among workers or maintaining sustainable conditions for workstations. In this work, we develop several Answer Set Programming encodings for solving SALBP-WS. The performed experiments showcase better scalability results compared to problem specific exact methods based on branch and bound technique. Additionally, we propose a new criterion that handles the workstation idle times hierarchically, as a measure of smoothness of an assignment.

Keywords: Answer set programming · Assembly line balancing · Workload smoothing.

1 Introduction

Many production facilities feature an assembly line to produce large quantities of standardized products. In such a facility the whole process of producing a product is usually broken down into various indivisible tasks that need to be performed in some predefined order. These tasks have to be completed by workstations (or people), which are organized sequentially along the assembly line. Moreover, a production facility needs to consider the overall production rate in order to meet various deadlines. Regarding this, the facility considers cycle time, the maximum amount of time allowed for each workstation to complete all of its assigned tasks. The main aim of the Assembly Line Balancing Problems (ALBP) is to assign tasks to workstations without violating the mentioned constraints. ALBP is an extensively studied topic in Operations and Production Research communities [2]. One important class of ALBPs is called Simple Assembly Line Balancing Problems (SALBP), where the assembly line is producing a single type of product. Furthermore, SALBP is partitioned into various problem types [12]. The feasibility problem (SALBP-F), for instance, searches a feasible assignment of tasks to workstations given a cycle time value and number of workstations.

[*] Copyright © 2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

Another variant of SALBP is the workload smoothing problem (SALBP-WS), which is NP-hard [1]. SALBP-WS is an optimization problem based on SALBP-F and aims at finding a feasible assignment such that resulting workloads of workstations are balanced as much as possible. This is important for establishing equity among workers or maintaining sustainable conditions for workstations. In this work, we are concentrating on the SALBP-WS.

Answer Set Programming (ASP) is a declarative problem solving paradigm with roots in Knowledge Representation and Reasoning (KRR). The simple but powerful language of ASP backed with highly efficient solver implementations make it applicable to various problems from broad range of domains [4, 6].

We developed a succinct ASP encoding of the SALBP-WS thanks to some powerful features of the ASP language like aggregates and optimization statements. To the best of our knowledge, this work is the first application of ASP to the problem. The resulting logic program can also be a foundation for other ALBPs. The experiments we have performed showcase better scalability results compared to problem specific SALBP-WS solvers.

Additionally, we propose a novel optimization criterion for the SALBP-WS to measure the smoothness of an assignment. Our motivation is to ease the burden on the ASP solver of large numbers appearing as optimization values in other criteria. The new *hierarchical idle times* (HIT) criterion aims to minimize idle times of workstations hierarchically. The use of HIT in our encoding leads to significant improvement in the solving performance. Apart from this, HIT can be useful in general for ALBPs.

In what follows, we formally define the SALBP-F and SALBP-WS. In Section 3, we describe our ASP encodings. Several experiments are performed using the openly available problem instances and the results of these are explained in Section 4. Finally, we conclude with discussion and future line of research.

2 Problem Definition

First, we define the feasibility version of the problem. Next, we cover the SABLP-WS when an optimization function based on various smoothness criteria is added on top of the feasibility problem.

2.1 The Simple Assembly Line Balancing Feasibility Problem (SALBP-F)

Consider n *tasks* and let $\mathcal{T}$ be the set $\{t_1, \ldots, t_n\}$ of all these tasks. For a given task t, $\mathbf{t}(t)$ gives the amount of time, which is named as *task time*, needed by any workstation for fully processing t. We assume all workstations are equally equipped and a task can be processed by any workstation. Considering a single-model assembly line with m *workstations*, let $\mathcal{S}$ be the set $\{s_1, \ldots, s_m\}$ of all workstations. Note that workstations are organized sequentially along the assembly line. Let $\mathbf{i}$ be the function that gives the order of a workstation in the line. The common setting adopted in open datasets of the literature is assigning positive integers as ids of workstations in a way that the one with 1 as an id is the first workstation in the assembly line. Following this setting, $\mathbf{i}(s_j) = j$. In a running assembly line an end product is produced at the end of each cycle. Let c be this *cycle time*. The main task of an assembly line balancing problem is to assign a task to a workstation. Let x_t^s be a decision variable such that $x_t^s = 1$ whenever task t is assigned to workstation s and $x_t^s = 0$ otherwise. The *workload* of a workstation is the total time needed for processing all tasks assigned to it. Considering the line balancing denoted by assignment decision variables, the workload time of a workstation $s \in \mathcal{S}$ is calculated by the following equation.

$$\mathbf{wl}(s) = \sum_{t \in \mathcal{T}} \mathbf{t}(t) \cdot x_t^s \tag{1}$$

Additionally, tasks have to be performed in some predefined order, which can be due to the nature of the product or the organizational constraints, such that no task can be performed unless all of its predecessors are complete. This condition and the fixed order of workstations in the assembly line constrains the assignment of tasks to workstations. To this end, we have an input directed acyclic graph $\mathcal{P} = (\mathcal{T}, E)$, which is called the *precedence graph*. The vertices are all the tasks and an edge $(u, v) \in E$ denotes that $\mathbf{i}(a) \leq \mathbf{i}(b)$, where $u, v \in \mathcal{T}$, $x_u^a = 1$ and $x_v^b = 1$. Recall that $\mathbf{i}(a)$ designates the order of a in the assembly line via an ordinal number.

Now, we can formally define SALBP-F. Given $\mathcal{T}$, $\mathcal{S}$, c and $\mathcal{P}$, the SALBP-F problem aims to find an assignment of tasks to workstations via setting $x_t^s = 1$ or 0 for all $t \in \mathcal{T}$ and $s \in \mathcal{S}$ while satisfying the following constraints.

$$\sum_{s \in \mathcal{S}} x_t^s = 1 \qquad (\forall t \in \mathcal{T}) \tag{2}$$

$$\sum_{t \in \mathcal{T}} \mathbf{t}(t) \cdot x_t^s \leq c \qquad (\forall s \in \mathcal{S}) \tag{3}$$

$$\sum_{a \in \mathcal{S}} \mathbf{i}(a) \cdot x_u^a \leq \sum_{b \in \mathcal{S}} \mathbf{i}(b) \cdot x_v^b \qquad (\forall u, v \in \mathcal{T}; (u, v) \in E) \tag{4}$$

The constraint (2) forces that each task must be assigned to a unique workstation. Additionally, the workload of a workstation cannot be greater than the cycle time of the assembly line. This is modeled by the constraint (3). Note that the left side of the inequality in (3) is the workload of s as given in (1). The constraint (4) makes sure that the precedence graph is respected in an assignment. Basically, for an edge (u, v) in this graph forming a precedence relation between u and v necessitates that task v is assigned to a workstation that the task u is also assigned to or to a one that comes later from the workstation that u is assigned to considering the order of workstations.

2.2 The Simple Assembly Line Workload Smoothing Problem (SALBP-WS)

The SALBP-WS is an optimization problem based on SALBP-F. It is also known as the Type III problem [5]. The SALBP-WS aims at finding an assignment such that resulting workloads of workstations are balanced as much as possible given the assignment satisfies the constraints of the feasibility variant SALBP-F.

An assignment is balanced for workstations when their workloads are equal or close to each other as much as possible. In order to achieve this optimization objective, various optimization criteria have been proposed. The two prominent ones are smoothness index and mean absolute deviation of workloads. The *smoothness index* (SI) value is calculated by the sum of squared differences between workstation workloads and the cycle time as shown in the equation below [1, 10].

$$SI = \sum_{s \in \mathcal{S}} (c - \mathbf{wl}(s))^2 \qquad \min SI \tag{5}$$

The *mean absolute deviation of workloads* (MAD) value is calculated by the sum of absolute deviations of workstation workloads from the average workload (i.e., the sum of all task times divided by the number of workstations) [11].

$$MAD = \sum_{s \in \mathcal{S}} \left| \mathbf{wl}(s) - \frac{\sum_{t \in \mathcal{T}} \mathbf{t}(t)}{|\mathcal{S}|} \right| \qquad \min MAD \tag{6}$$

The SALBP-WS utilizing SI as the optimization criterion aims to minimize the SI value of a feasible solution. Hence, its definition is the addition of the whole objective (5) to the setting of SALBP-F. In case MAD is utilized as the criterion in SALBP-WS, the objective (6) is added.

The following running example is selected from [2].

Example 1. In an single-model assembly line there are 5 workstations; $\mathcal{S} = \{s_1, \ldots, s_5\}$ and their order is given by $\mathbf{i}(s_x) = x$. The production needs 10 tasks $\mathcal{T} = \{1, \ldots, 10\}$, where the precedence graph $\mathcal{P}$ is depicted in Figure 1. The task times are also given in upper right corners of task nodes in $\mathcal{P}$ (for instance, $\mathbf{t}(8) = 2$).

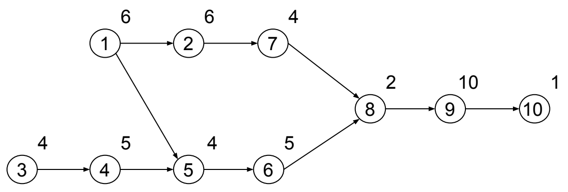

Fig. 1. Precedence graph $\mathcal{P}$

Considering the assembly line configuration given in Example 1, a solution for the SALBP-F is shown diagrammatically in Figure 2. The solution assigns $\{3, 4\}$, $\{1\}$, $\{2, 5\}$, $\{6, 7, 8\}$ and $\{9, 10\}$ to workstations s_1, s_2, s_3, s_4 and s_5, respectively. Consequently, the workload times $\mathbf{wl}(s_4) = 11$ and $\mathbf{wl}(s_5) = 11$ (i.e., they have 0 idle times), and for the other workstations $\mathbf{wl}(s_1) = 9$, $\mathbf{wl}(s_2) = 6$ and $\mathbf{wl}(s_3) = 10$ (i.e., they have 2, 5 and 1 idle times, respectively). Note that SI value

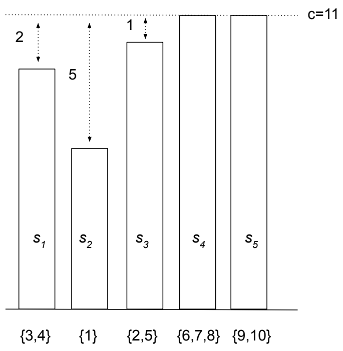

Fig. 2. A feasible solution with $SI = 30$ and $MAD = 7.6$

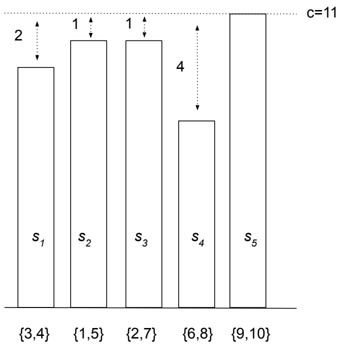

Fig. 3. A SALBP-WS solution with $SI = 22$ and $MAD = 5.6$

of the feasible solution shown in Figure 2 is $2^2+5^2+1^2+0^2+0^2 = 30$. Additionally, its MAD value is $|9-9.4|+|6-9.4|+|10-9.4|+|11-9.4|+|11-9.4| = 7.6$. It is not a SI optimal solution. Figure 3 depicts a solution for the SALBP-WS problem of the assembly line configuration given in Example 1. Note that the workstations have 2, 1, 1, 4 and 0 idle times, respectively. The SI value of this optimal solution is $2^2 + 1^2 + 1^2 + 4^2 + 0^2 = 22$. Regarding the MAD criterion, the solution in Figure 3 has $|9-9.4| + |10-9.4| + |10-9.4| + |7-9.4| + |11-9.4| = 5.6$ as the MAD value. Actually, it is also an optimal solution for the SALBP-WS problem when MAD is used as the criterion.

2.3 A New Criterion for Workload Smoothness

In our initial experiments we have observed that large numbers as optimization values of SI and MAD criteria (especially the quadratic value in (5)), put a strain on the ASP solver. In order to improve solving performance we have developed a novel criterion for measuring workload smoothness. The *hierarchical idle times* (HIT) value of an assignment is a vector of numbers of workstations grouped by hierarchically decreasing idle times starting from the maximum idle time.

$$HIT = \langle a_m, \ldots, a_1 \rangle \qquad \text{s.t. } a_i = |\,\{s_x | c - \mathbf{wl}(s_x) = i\}\,|, i \leq c \tag{7}$$

The HIT value of the feasible assignment in Figure 2 is $\langle 1, 0, 0, 1, 1 \rangle$. Note that there is 1 workstation with idle time of 5 (s_2), 1 workstation with idle time of 2 (s_1) and 1 workstation with idle time of 1 (s_3). The 0 values in the vector correspond to the number of workstations with idle times 4 and 3. We also do not consider the number of workstations with no idle times (for instance, the feasible assignment has 2 workstations with 0 idle time). Additionally, 0 values for the number of workstations with idle times $5 < i \leq c$ are skipped.

When we want to utilize the HIT criterion, the SALBP-WS problem is modeled as a multi-objective lexicographical optimization problem unlike the cases in SI and MAD. Note that ASP has the capacity of modeling such multi-objective optimization problems. We try to minimize the HIT value of an assignment in a lexicographical way such that along the optimization minimizing the number of workstations with i idle times is given more priority than minimizing the number of workstations with j idle times where $j < i$. Formally, an assignment with a HIT value $\langle a_1, \ldots, a_i, \ldots, a_x \rangle$, is better than an assignment with a HIT value $\langle b_1, \ldots, b_i, \ldots, b_x \rangle$ given that $a_k = b_k$ for all $1 \leq k < i$ and $a_i < b_i$.

Hence, SALBP-WS can also be modeled by adding the following optimization objective to the setting of SALBP-F.

$$\text{lexmin } HIT \tag{8}$$

For Example 1, the solution shown in Figure 3 is also an optimal solution when HIT is used as the criterion. Note that the HIT vector of this optimal solution is $\langle 1, 0, 1, 2 \rangle$. In order to easily compare with the HIT value of the solution shown in Figure 2, one can also see $\langle 0, 1, 0, 1, 2 \rangle$ as the HIT value of the optimal solution since there are 0 workstations with an idle time of 5.

3 Encoding SALBP-WS via ASP

In this section, we first encode SALBP instances. Then, we explain the encoding of the SALBP-F variant. With the addition of encodings for smoothness criteria on top of the SALBP-F encoding, we complete our SALBP-WS encoding. For the syntax and semantics of the language used in encodings, one may refer to [3]. Additionally, detailed information about ASP based problem solving paradigm and the ASP solver *clingo* can be found in [7].

3.1 Encoding SALBP Instances

The Listing 1.1 gives an instance encoding for Example 1 given in Section 2.

```
1   #const cycletime=11.

3   task(1,6).  task(2,6).  task(3,4).  task(4,5).  task(5,4).
4   task(6,5).  task(7,4).  task(8,2).  task(9,10).  task(10,1).

6   precedence(1,2).  precedence(1,5).  precedence(3,4).  precedence(4,5).
7   precedence(5,6).  precedence(2,7).  precedence(7,8).  precedence(6,8).
8   precedence(8,9).  precedence(9,10).

10  workstation(1).  workstation(2).  workstation(3).  workstation(4).
11  workstation(5).
```

Listing 1.1. Encoding of the SALBP instance given in Example 1

The cycle time $c = 11$ is represented by the `cycletime` constant in Line 1. The decision of making c an ASP constant is to facilitate easy testing by trying different cycle time values in the *clingo* command line. The tasks and workstations in an instance are represented by ASP facts `task/2` and `workstation/1`, respectively, in the following form: `task(`t`,`i`)` s.t. $t \in \mathcal{T}, i = \mathbf{t}(t)$ and `workstation(`$\mathbf{i}(s)$`)` s.t. $s \in \mathcal{S}$. For the sake of simplicity in encodings and the common way of naming in open datasets, we use the order $\mathbf{i}(S)$ of a workstation S as its identifier. The tasks and workstations of Example 1 are listed in Lines 3–4 and Lines 10–11, respectively. The precedence graph of an instance is represented by `precedence/2` facts of form `precedence(`u`,`v`)` s.t. (u, v) is an edge of $\mathcal{P}$. The facts in Lines 6–8 represent the precedence graph in Figure 1.

Although the instance format is developed specifically for SALBP-F and SALBP-WS instances, it can also be utilized for other SALBP variants.

3.2 Encoding SALBP-F

Having represented a SALBP instance in ASP, we now encode the constraints of the SALBP-F. The assignment of a task to a workstation is represented by instances of the binary predicate `assignment/2` in the form `assignment(`s`,`t`)` s.t. $s \in \mathcal{S}, t \in \mathcal{T}$. Given a task t and a workstation s, if the atom `assignment(`s`,`t`)` is in an answer set (or, if it is true), the SALBP-F solution corresponding to the answer set must have $x_t^s = 1$. Otherwise, the solution must have $x_t^s = 0$ (i.e., if the atom is not in the answer set).

```
1 {assignment(W,T) : workstation(W)} = 1 :- task(T,C).

3 :- #sum{C,T: assignment(W,T), task(T,C)} > cycletime,
4    workstation(W).

6 :- precedence(X,Y), assignment(W1,X), assignment(W2,Y), W1>W2.

8 #show assignment/2.
```

Listing 1.2. Encoding of the SALBP-F

The rules in Listing 1.2 comprise the ASP encoding of SALBP-F. Based on the generate-and-test technique, a commonly used encoding technique in ASP [7], the choice rule in Line 1 generates the search space based on x_t^s decision variables of SALBP-F. The same rule also represents the constraint (2) of SALBP-F. The equality condition in the rule head forces that a task must be assigned to a unique workstation. The constraint rule in Lines 3–4 represents the constraint (3). The `#sum` aggregate in the rule, basically, computes the workload $\mathbf{wl}(s)$ of any workstation s by summing up task times of all tasks assigned to s in a model of the logic program. Being a constraint rule and having the condition '`> cycletime`', it encodes that whenever a model of the program leads to a workstation with a workload greater than the cycle time, that model cannot be an answer set (so, that model is eliminated). Similarly, the constraint rule in Line 6 represents the constraint (4) about the precedence graph. Given an edge (x, y) of $\mathcal{P}$ (represented by an instance of `precedence(X,Y)` in Line 6), the constraint rule eliminates a model where tasks x and y are assigned to workstations $w1$ and $w2$ (instances of variables `W1` and `W2`), respectively, such that the order of $w1$ is later than the order of $w2$ (i.e., $\mathbf{i}(w1) > \mathbf{i}(w2)$). Note that in such a case the precedence constraint is not respected. The `#show` statement in Line 8 restricts the answer set output of *clingo* to only instances of `assignment/2` predicate. In this way, one can easily compile a SALBP-F solution from an answer set of the program.

Let Π be the encoding of SALBP-F composed of rules in Listing 1.2. When we run *clingo* with the program Π and the instance rules in Listing 1.1 as input, one answer set we get is the following one.

```
assignment(2,1) assignment(3,2) assignment(3,5) assignment(1,4)
assignment(4,6) assignment(4,7) assignment(4,8) assignment(5,9)
assignment(1,3) assignment(5,10)
```

Considering the semantics of the `assignmnet/2` predicate, this answer set represents the SALBP-F solution shown in Figure 2.

3.3 Encoding Smoothness Criteria and SALBP-WS

Encoding SI. The *clingo* solver has the capacity of solving optimization problems [8]. For such problems, it features `#minimize` and `#maximize` statements. Considering this capacity, let us encode the SI criterion in ASP to be used modularly as an addition to the SALBP-F encoding. The rules in Listing 1.3 represent the SI criterion and respective optimization objective function given in (5).

```
1 workload(W,L) :- L=#sum{C,T :assignment(W,T), task(T,C)},
2                  workstation(W), cycletime>=L.

4 #minimize{(cycletime-L)*(cycletime-L),W : workload(W,L)}.
```

Listing 1.3. Encoding of the SI criterion for SALBP-WS

With the help of the `#sum` aggregate, the rule in Lines 1–2 finds the workload of each workstation. An atom `workload(`w`,`l`)`, which is generated by the rule as an instance of the `workload/2` predicate, represents that the workload of workstation w is l, i.e., $\mathbf{wl}(w) = l$. A careful reader may notice the '`cycletime>=L`' condition in Line 2 is not necessary considering the SALBP-F constraint in Lines 3–4 of Listing 1.2. Although it is redundant w.r.t. the semantics of the program, it improves the grounding performance of *clingo* by avoiding the generation of many futile ground instances of the respective rule (viz., ground rules with head atoms such as `workload(4,33)` or `workload(5,42)` where workload is over the cycle time). Although the heads of such futile rule instances will trivially not be included in any answer set, they not only degrade grounding performance of *clingo*, but also hinder the solving performance. The `#minimize` statement in Line 4 represents the SI objective function in (5).

Let Π_{SI} be the SALBP-WS encoding composed of the SALBP-F encoding in Listing 1.2 and the encoding of SI criterion in Listing 1.3. When we run *clingo* with Π_{SI} and the instance in Listing 1.1 as input, we get the following output.

```
Answer: 1
assignment(2,1) assignment(3,2) assignment(3,5) assignment(1,4)
assignment(4,6) assignment(4,7) assignment(4,8) assignment(5,9)
assignment(1,3) assignment(5,10)
Optimization: 30
Answer: 2
assignment(2,1) assignment(3,2) assignment(2,5) assignment(1,4)
assignment(4,6) assignment(3,7) assignment(4,8) assignment(5,9)
assignment(1,3) assignment(5,10)
```

```
Optimization: 22
OPTIMUM FOUND
```

Notice that *clingo* finds a feasible solution and tries to improve that solution incrementally w.r.t. the SI value. The `OPTIMUM FOUND` message in the output designates that *clingo* has managed to prove the last answer set found is indeed an optimal one. Actually, that answer set corresponds to the optimal SALBP-WS solution given in Figure 3.

Encoding MAD. Similar to the encoding of the SI criterion, we can encode the MAD criterion in ASP. The ideal average workload of each workstation calculated in (6), however, can be a real number and *clingo* does not support programs with real numbers. This hurdle can be overcome by the following Lemma.

Lemma 1. *The objective of minimizing of the MAD value in (6) is equivalent to the objective*

$$min \sum_{s\in\mathcal{S}} \left| \mathbf{wl}(s)|\mathcal{S}| - \sum_{t\in\mathcal{T}} \mathbf{t}(t) \right| \ . \tag{9}$$

Proof.

$$\begin{aligned} MAD &= \sum_{s\in\mathcal{S}} \left| \mathbf{wl}(s) - \frac{\sum_{t\in\mathcal{T}} \mathbf{t}(t)}{|\mathcal{S}|} \right| = \sum_{s\in\mathcal{S}} \left| \frac{\mathbf{wl}(s)|\mathcal{S}| - \sum_{t\in\mathcal{T}} \mathbf{t}(t)}{|\mathcal{S}|} \right| \\ &= \frac{1}{|\mathcal{S}|} \sum_{s\in\mathcal{S}} \left| \mathbf{wl}(s)|\mathcal{S}| - \sum_{t\in\mathcal{T}} \mathbf{t}(t) \right| \end{aligned}$$

Hence, the objective of min $MAD =$ min $\sum_{s\in\mathcal{S}} \left|\mathbf{wl}(s)|\mathcal{S}| - \sum_{t\in\mathcal{T}} \mathbf{t}(t)\right|$. □

One may get tempted to minimize $\sum_{s\in\mathcal{S}} \mathbf{wl}(s)$ as the optimization objective, considering other parts of the equation (9) is constant. However, this is simply incorrect for solving SALBP-WS, since $\sum_{s\in\mathcal{S}} \mathbf{wl}(s)$ is equal to the total task time and is fixed for a problem instance. A similar issue also appears for the SI criterion, where minimizing $\sum_{s\in\mathcal{S}} (c - \mathbf{wl}(s))$ (droping the square operation) is incorrect since it is a fixed value for an instance.

Using Lemma 1, the rules in Listing 1.4 encode the MAD criterion and the respective objective function.

```
1  totaltask(TT) :- TT = #sum{C,T : task(T,C)}.
2  numworkstation(NW) :- NW = #count{W : workstation(W)}.

4  #minimize{|(L*N)-T|,W : workload(W,L), totaltask(T),
5                                numworkstation(N)}.
```

Listing 1.4. Encoding of the MAD criterion for SALBP-WS

The rules in Lines 1–2 calculate the number of workstations and total task time. The `#minimize` statement in Line 4–5 represents the objective function (9).

Let Π_{MAD} be a SALBP-WS encoding composed of Listing 1.4, Listing 1.2 and Lines 1–2 of Listing 1.3 (the definition of `workload` predicate). Given the program Π_{MAD} and the instance Listing 1.1, *clingo* finds an optimal answer set that corresponds to the optimal SALBP-WS solution given in Figure 3.

Encoding HIT. The rules in Listing 1.5 encodes the HIT criterion (7) and the respective objective function (8). First, the rule in Lines 1–2 finds idle time of each workstation similar to the rule (Lines 1–2 of Listing 1.3) finding workloads. An atom of the form `idletime(`w`,`i`)` in an answer set means the idle time of workstation w is i, i.e., $c - \mathbf{wl}(w) = i$. Next, rules in Lines 4–6 generate the lexicographical levels as mentioned in the HIT value. The maximum idle time is found (Line 4) and levels from this maximum level until the level 1 (inclusive) are generated (Lines 5–6). Finally, the HIT objective in (8) is encoded with the help of optimization statement in Line 8. The statement commands *clingo* to minimize for each level the number of workstations having idle times of that level. *clingo* implements multi-criteria lexicographical optimization via priorty levels in weights. Summing up the weights given as `1` in Line 8 corresponds to the number of workstations having idle time of `L` and the `@L` priority levels define the lexicographical nature of HIT.

```
1 idletime(W,cycletime-L) :- workstation(W), cycletime>=L,
2                L = #sum{C,T : assignment(W,T), task(T,C)}.

4 max(M) :- M = #max{ L: idletime(W,L) }.
5 levels(M) :- max(M), M>0.
6 levels(L-1) :- levels(L), L>1.

8 #minimize{ 1@L,W:  idletime(W,L), levels(L) }.
```

Listing 1.5. Encoding of the HIT criterion for SALBP-WS

Let Π_{HIT} be the SALBP-WS encoding composed of Listing 1.2 and Listing 1.5. Running *clingo* with Π_{HIT} and the problem instance Listing 1.1, we can solve the SALBP-WS for Example 1 and get the optimal solution given in Figure 3.

4 Experimental Evaluation

We have conducted several experiments to analyse performance of the ASP encodings Π_{SI}, Π_{MAD} and Π_{HIT} defined in Section 3.3. To this end, we used several benchmarks from a dataset mentioned in [13] and openly available from `https://assembly-line-balancing.de/`. All the experiments have been performed in a Linux machine with Intel E5-2630@2.20GHz CPU. For computing optimal answer sets, we have used *clingo* version 5.4.0 with 6GB as the memory limit and 600 seconds as the time limit.

The results of the ASP encodings are shown in Table 1. Instances of the experiment are partitioned into groups with a fixed number of tasks and a fixed precedence graph. For instance, all instances of the Mitchell group have 21 tasks (shown by the $|\mathcal{T}|$ column) and the same precedence graph. They may vary in the number of workstations (shown by the $|\mathcal{S}|$ column) and the cycle time (shown by the c column). The results of each encoding are under Π_{SI}, Π_{MAD} and Π_{HIT}

Table 1. Performance results of the Π_{SI}, Π_{MAD} and Π_{HIT} ASP encodings.

				Π_{SI}			Π_{MAD}			Π_{HIT}		
Problem	$\|\mathcal{T}\|$	$\|\mathcal{S}\|$	c	SI	MAD	Time	SI	MAD	Time	SI	MAD	Time
Mitchell	21	3	35	0	0.0	0.02	0	0.0	0.01	0	0.0	0.02
	21	3	39	48	0.0	0.02	48	0.0	0.02	48	0.0	0.02
	21	5	21	0	0.0	0.02	0	0.0	0.02	0	0.0	0.02
	21	5	26	125	0.0	0.05	125	0.0	0.02	125	0.0	0.02
	21	8	14	9	3.5	0.02	9	3.5	0.02	9	3.5	0.02
	21	8	15	31	3.5	0.03	31	3.5	0.03	31	3.5	0.03
Roszieg	25	4	32	5	3.0	0.02	5	3.0	0.02	5	3.0	0.02
	25	6	21	1	1.7	0.03	1	1.7	0.03	1	1.7	0.03
	25	6	25	105	1.7	0.2	105	1.7	0.06	105	1.7	0.05
	25	8	16	5	4.5	0.03	5	4.5	0.03	5	4.5	0.03
	25	8	18	49	4.5	0.23	49	4.5	0.08	49	4.5	0.1
	25	10	14	59	12.0	0.08	59	12.0	0.07	59	12.0	0.08
Sawyer	30	5	75	521	1.6	333.87	521	1.6	0.12	521	1.6	1.24
	30	7	47	11	5.7	0.19	11	5.7	0.24	11	5.7	0.21
	30	7	54	420	3.4	–	420	3.4	0.24	420	3.4	3.95
	30	8	41	4	4.0	0.32	4	4.0	0.05	4	4.0	0.36
	30	10	36	136	6.8	–	136	6.8	1.5	136	6.8	3.49
	30	11	33	149	8.4	–	149	8.4	3.12	149	8.4	2.64
	30	12	30	116	6.0	444.01	116	6.0	0.84	116	6.0	1.37
	30	13	27	61	5.5	43.93	61	5.5	1.87	61	5.5	0.63
	30	14	25	60	10.6	19.58	60	9.1	3.16	60	9.1	0.84
Gunther	35	7	81	1186	24.0	37.54	1198	24.0	2.21	1186	24.0	0.28
	35	8	69	615	9.0	94.03	615	9.0	0.57	615	9.0	0.37
	35	9	54	3	4.0	0.25	3	4.0	0.25	3	4.0	0.19
	35	9	61	486	4.0	–	486	4.0	0.59	486	4.0	0.39
	35	11	49	310	11.1	166.38	310	11.1	1.13	310	11.1	0.46
	35	12	44	205	17.5	1.03	205	17.5	0.83	205	17.5	0.32
	35	14	41	1519	61.0	26.8	1545	61.0	4.51	1519	61.0	0.79
Lutz3	89	12	150	2152	32.0	–	2032	4.0	39.08	2032	4.0	–
	89	13	137	1625	40.6	–	1459	11.5	157.76	1521	20.3	–
	89	14	118	10	8.0	3.7	10	8.0	6.26	10	8.0	4.67
	89	14	127	1410	34.6	–	1288	8.0	96.82	1288	8.0	–
	89	15	110	8	8.0	9.95	8	8.0	14.88	8	8.0	9.28
	89	17	103	763	32.5	–	709	18.5	462.58	721	19.3	–
	89	18	97	664	33.3	–	638	20.7	538.4	640	23.3	–
	89	19	92	688	38.4	–	780	36.4	–	616	25.6	–
	89	20	87	638	37.6	–	648	36.0	79.7	648	38.8	17.56
	89	21	83	573	32.9	–	553	24.3	–	553	24.3	–
	89	22	79	468	31.6	–	454	25.6	120.75	484	37.5	49.97
	89	23	75	347	32.4	–	349	27.6	161.28	339	28.4	18.43

columns. For each encoding, we report the SI (shown by the SI column) and MAD (shown by the MAD column) values of the best solution found within the time limit. By doing so, we aim to investigate whether one can select any criterion without significant deviation in terms of quality or not (this is discussed further below). If *clingo* has proved that the best found solution is indeed an optimal one we report the time used (shown by the Time column) and otherwise (i.e., *clingo* does not find an optimal solution within 600 seconds), we report $-$ as the time value.[1]

One main result is that when the number of tasks increases with a usual consecutive increase in cycle time, instances become much harder to solve. The Π_{SI} encoding struggles earlier than the two other ones and in total finds less number of optimal solutions. The Π_{MAD} and Π_{HIT} encodings scale better. We think *clingo* has difficulty in minimizing big quadratic numbers in (5) for Π_{SI}, while it shows better performance in minimizing linear counts in (7) for Π_{HIT}. Better performance of Π_{MAD} can be explained with its less sensitiveness to cycle time. Note that the MAD criterion in (6) (or (9)) does not involve cycle time.

Another important result is about the SI and MAD values of the optimal solutions. Considering the instances where any two encodings find an optimal solution, the corresponding SI and MAD values of these solutions are the same except a few cases. For instance, Gunther 35, 7, 81 is one exception where the optimal solution found by Π_{MAD} has SI value of 1198, which is slightly more than 1186 the SI value of the optimal solutions found by Π_{SI} and Π_{HIT}. This consistent agreement on SI and MAD values of all encodings shows that one can select any of the encodings or optimization criteria. This is a significant observation since the recent studies in the research community focus only on SI, while another criterion may improve computation performance without any significant deviation in terms of quality. We have experienced exactly this case for our encodings where Π_{MAD} and Π_{HIT} perform clearly better than Π_{SI}.

Unlike the other SALBP types, studies on SALBP-WS are scarce. Being an intractable problem, most of the efforts to solve the SALBP-WS focus on heuristic methods [11, 10, 5]. In [1], however, an exact method of solving SALBP-WS with the SI criterion is studied. Basically, they have implemented a problem specific search algorithm based on branch and bound technique. Neither their implementation nor the instances they used are openly available.[2] However, it is explicitly stated in [1] that their algorithm can solve instances with up to 29 tasks. Our way of solving SALBP-WS via ASP is also an exact method and the results in Table 1 show that our encodings scale better w.r.t. the number of tasks.

Another exact method for solving SALBP-WS is [9]. They have also implemented a branch and bound based algorithm that utilizes heuristics for computing lower and upper bounds. Their empirical studies uses 117 instances from the same ALBP dataset website we used. We have also tested our ASP encodings with these 117 instances and the results are reported in Table 2. The number of

[1] We observed that grounding times of clingo are not significant as solving times.

[2] Although, they use some of the openly available precedence graphs, they have generated task times randomly.

optimal solutions found for each problem group are listed with an aggregate on the last row. Although Π_{SI} is not performing well (an optimal solution is found for 32 instances) as the problem specific implementation of [9] (67 instances), Π_{HIT} finds almost the same number of optimal solutions (66 instances) and Π_{MAD} finds more optimal solutions (72 instances) within the time limit.

Table 2. The number of optimal solutions found for the 117 instances from [9]. 600 seconds is used as a time limit for the ASP encodings and 3600 seconds is used in [9].

Problem	$\lvert\mathcal{T}\rvert$	#I	from [9]	Π_{SI}	Π_{MAD}	Π_{HIT}
Mitchell	21	9	9	9	9	9
Roszieg	25	9	9	9	9	9
Heskia	28	9	8	1	8	1
Buxey	29	9	9	2	9	9
Sawyer	30	9	7	2	9	9
Lutz1	32	9	9	4	9	9
Gunther	35	9	6	3	8	9
Kilbrid	45	9	2	1	3	1
Hahn	53	9	6	1	7	9
Warnecke	58	9	0	0	1	1
Tonge	70	9	2	0	0	0
Wee-mag	75	9	0	0	0	0
Arc83	83	9	0	0	0	0
$\sum$		117	67 (50)	32 (85)	72 (45)	66 (51)

5 Discussion

In this paper, we have encoded the SALBP-WS in ASP. This problem aims at balancing the workloads evenly among workstations in an assembly line. We have performed various experiments and they showcase better scalability results compared to problem specific SALBP-WS solvers.

The modular ASP encodings make it possible for us to try various criteria to quantify how balanced the SALBP assignment is. This issue is especially important for SALBP-WS solvers using problem specific search algorithms since they heavily depend on one optimization criterion (not just due to calculation of the optimization function, but also the need for theoretical bound analysis for improving solving performance) and supporting another one is not as easy as in the case of ASP.

Most of the recent related work on SALBP-WS rely on the SI criterion [10, 1, 9]. Our experiments show other criteria like MAD can also be used without significant deviation in assignment quality. This can even improve the computational performance of the system like in the case of our encodings.

Apart from criteria SI and MAD, we proposed a novel criterion based on idle times of workstations. Using the new HIT criterion, SALBP-WS problem can be modeled as a lexicographical multi-objective optimization problem. This new criterion can also be beneficial for other work related to ALBPs.

One important future line of research is to benefit from other SALBP-WS solvers that compute lower and upper bounds [1, 9] for the problem instance in ASP encodings.

References

1. Azizoğlu, M., İmat, S.: Workload smoothing in simple assembly line balancing. Computers & Operations Research **89**, 51–57 (2018). https://doi.org/10.1016/j.cor.2017.08.006
2. Becker, C., Scholl, A.: A survey on problems and methods in generalized assembly line balancing. European Journal of Operational Research **168**(3), 694–715 (2006). https://doi.org/10.1016/j.ejor.2004.07.023
3. Calimeri, F., Faber, W., Gebser, M., Ianni, G., Kaminski, R., Krennwallner, T., Leone, N., Maratea, M., Ricca, F., Schaub, T.: Asp-core-2 input language format. Theory Pract. Log. Program. **20**(2), 294–309 (2020)
4. Erdem, E., Gelfond, M., Leone, N.: Applications of Answer Set Programming. AI Magazine **37**(3), 53–68 (2016). https://doi.org/10.1609/aimag.v37i3.2678
5. Eswaramoorthi, M., Kathiresan, G.R., Jayasudhan, T.J., Prasad, P.S.S., Mohanram, P.V.: Flow index based line balancing: a tool to improve the leanness of assembly line design. International Journal of Production Research **50**(12), 3345–3358 (2012). https://doi.org/10.1080/00207543.2011.575895
6. Falkner, A., Friedrich, G., Schekotihin, K., Taupe, R., Teppan, E.C.: Industrial Applications of Answer Set Programming. KI - Künstliche Intelligenz **32**(2), 165–176 (2018). https://doi.org/10.1007/s13218-018-0548-6
7. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2012)
8. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo= ASP+ control: Preliminary report. arXiv preprint arXiv:1405.3694 (2014), `http://arxiv.org/abs/1405.3694`
9. Hazır, Ö., Agi, M.A., Guérin, J.: An efficient branch and bound algorithm for smoothing the workloads on simple assembly lines. International Journal of Production Research pp. 1–18 (2019). https://doi.org/10.1080/00207543.2019.1701208
10. Mozdgir, A., Mahdavi, I., Badeleh, I.S., Solimanpur, M.: Using the Taguchi method to optimize the differential evolution algorithm parameters for minimizing the workload smoothness index in simple assembly line balancing. Mathematical and Computer Modelling **57**(1), 137–151 (2013). https://doi.org/10.1016/j.mcm.2011.06.056
11. Rachamadugu, R., Talbot, B.: Improving the equality of workload assignments in assembly lines. International Journal of Production Research **29**(3), 619–633 (1991). https://doi.org/10.1080/00207549108930092
12. Scholl, A.: Balancing and Sequencing of Assembly Lines. Contributions to Management Science, Physica-Verlag Heidelberg, 2 edn. (1999)
13. Scholl, A., Boysen, N., Fliedner, M.: The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics. OR Spectrum **35**(1), 291–320 (2013). https://doi.org/10.1007/s00291-011-0265-0