Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 48 (2020) 71–81

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

48th SME North American Manufacturing Research Conference, NAMRC 48 (Cancelled due to COVID-19)

A Case Study in Line Balancing and Simulation

I. Ozan Yilmazlar[a]*, Adarsh Jeyes[a], Alexis Fiore[a], Apurva Patel[b], Chelsea Spence[a], Chase Wentzky[b], Nicole Zero[b], Mary E. Kurz[a], Joshua D. Summers[b], Kevin M. Taaffe[a]

[a]*Department of Industrial Engineering, Clemson University, Clemson, SC, 29634, USA*
[b]*Department of Mechanical Engineering, Clemson University, Clemson, SC, 29634, USA*

* Corresponding author. Tel.: +1-864-375-8007 ; fax: +1-864-656-0795. *E-mail address:* iyilmaz@clemson.edu

Abstract

Assembly line balancing (ALB) allocates individual tasks to work stations while respecting the physical, safety, and quality constraints. Two-sided assembly lines are generally used in the production of medium or large-sized products (e.g. automotive, household appliance). We considered several characteristics including zoning constraints, the task to task relationships, tooling and station dependent constraints to offer the real-world environment. The most common two objectives for the ALB are minimizing the number of workers (type-1) and minimizing the cycle time (type-2). This article presents an integer programming formulation for both type-1 and type-2 ALB problems and metaheuristics to solve this complex problem. Even if ALB gives better results than the current line balance that our industry partner applied, it cannot be guaranteed that the amount of daily production will increase due to randomness in the line. We simulate the proposed line balances to provide a testing platform for line balancing results and to help identify inefficiencies and bottlenecks in the system.

© 2020 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)
Peer-review under responsibility of the Scientific Committee of the NAMRI/SME.

Keywords: Assembly Line Balancing; Integer Programming; Ranked Positional Weight; Hill Climbing; Simulation

1. Motivation

Assembly line balancing (ALB) is a production efficiency improvement strategy that assigns predetermined tasks to work stations while respecting the precedence order and problem-specific constraints. The main objective of ALB is to distribute workload evenly among workers and to increase the output rate. Some mandatory steps to achieve this objective are as follows:

- Determine tasks and task properties
- Determine the precedence relationship among tasks
- Estimate (observe) task times
- Assign tasks to the stations
- Calculate efficiency

We assign tasks with one of two points of view: either minimize the number of workers subject to the amount of time allowed per station (cycle time) or minimize the cycle time subject to the number of workers available.

Mid-sized products like major household appliances are generally produced in two-sided assembly lines. The two-sided assembly line can perform tasks on both sides where there are sequential stations. Unlike multi-manned assembly lines that produce large-sized products, such as cars, two-sided assembly lines allow only one person to work on either side of the line. Parallel stations facing each other are called mated-stations.

The analytical statement of ALB was first introduced by Bryton (1954) [1] and there has been an enormous increase in the number of articles on this subject ever since. As the competition increased among the companies and they attempt to cut costs, manufacturers that do not apply ALB become unable to survive. Since ALB is so important for mass production, various mathematical formulation and exact solution methods have been presented over the last decades. It is well-known that ALB is an NP-hard problem as it is a reduction of the partition problem [2]. Because it is often difficult to find an optimal solution, especially for larger data sets, many heuristic and metaheuristic methods have been developed to solve the ALB problem (ALBP).

2351-9789 © 2020 The Authors. Published by Elsevier B.V.
This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)
Peer-review under responsibility of the Scientific Committee of the NAMRI/SME.
10.1016/j.promfg.2020.05.076

This project is intended to provide a method for ALB for a company making products like washing machines. The current techniques, as described in the literature review, may be inappropriate for the case study because the exact scenario we have is not represented. We investigate the appropriateness of these approaches.

We present an integer programming formulation and a hill-climbing algorithm to solve two-sided ALBP (TALBP) with real-world constraints, such as zoning, tooling, adjacency, and station dependent constraints. These supplementary constraints are explained in detail in Section 3. Finally, we present a simulation model to provide information about the future possible reactions of the production system against various situations, according to arranged scenarios.

The remaining of this paper organized as follows. Related literature is given in Section 2. We define the problem environment and real-world constraints in Section 3. An integer programming formulation and a metaheuristic algorithm are proposed in Section 4 and Section 5, respectively. Section 6 involves simulation modeling. The results obtained with this project presented in Section 7. Lastly, the discussion about the process of problem solving and implementation is shared in Section 8.

2. Literature Review

The very first study that might be considered as line balancing was in Bryton's MS thesis but the first published study came from Salveson who formulated a simple assembly line balancing problem (SALBP) [3]. SALBP is based on a set of limitations [4]:

1. Mass-production of one homogeneous product.
2. All tasks are processed in a predetermined mode.
3. Paced line with fixed common cycle time according to market demand.
4. The line is considered to be serial with no feeder lines or parallel elements.
5. The processing sequence of tasks is subject to precedence restrictions.
6. Deterministic task times.
7. No assignment restrictions of tasks besides precedence constraints.
8. No task can be split among two or more stations.
9. All stations are equally equipped with concerning machines and workers.

SALB problems differ according to objectives. Scholl categorized SALB problems into four categories [5]:

- SALBP-1 minimizes the number of stations for a given cycle time (type-1 objective).
- SALBP-2 minimizes the cycle time for a given number of stations (type-2 objective).
- SALBP-E minimizes the cycle time and the number of stations (minimizes the line efficiency).
- SALBP-F finds a balance for a given number of stations and a given cycle time.

Limitations (7) and (9) do not hold in our problem. Variations of these limitations are called as general ALB (gALB). Real-life problems have additional restrictions to the precedence constraints like tooling, zoning, worker skill, resource, and equipment. Sivasankaran and Shahabudeen distinguished the ALB problems based on three main features to present a well-structured review [6]:

- Number of models produced in the line
  - o Single model
  - o Multi model/Mixed model
- The variation of task times
  - o Deterministic task times
  - o Stochastic task times
- The variation of flow
  - o Straight type
  - o U type

Our problem is considered as a single model with deterministic task times and straight type ALB with both type-1 and type-2 objectives, thus we highly focus on this type of problem in the literature. Since the ALBP is an NP-hard problem [7], a large number of exact and heuristic algorithms have been proposed to solve the ALB problems in the literature.

The first mathematical formulation of SALBP, introduced by Bowman, used linear programming [8]. White presented a modified version of Bowman's formulation [9]. Thangavelu and Shetty presented a revised version of the Bowman-White zero-one integer programming formulation so that certain steps in Geoffrion's 0-1 integer programming algorithm can be simplified or eliminated to solve SALBP-1 [10]. The historical development of the SALBP mathematical models is given by Salama et al. [11]. Exact solution methods for SALBP-1 can be separated into branch and bound approaches and dynamic programming processes [12]. In the literature, there are many studies that applied branch and bound methods to find an exact solution for the applied SALBP [13–20]. Various dynamic programming methods have been used to solve SALBP, especially with stochastic task times [21–25].

Heuristic solution methods for SALBP mostly depend on ranked positional weight (RPW) algorithms [26]. The RPW method takes into account all task times and the precedence relationship. It is designed for fixed cycle time problems (SALBP-1). The first step is to draw a precedence diagram. Then, sort the tasks according to the positional weight, which is equal to the task duration plus the total duration of all of its successor tasks. Once the tasks are ranked in descending order of positional weight, assign the tasks to the first station as the cycle time allows. Then continue with the next station by following precedence constraints. Fathi et al. compared 20 different heuristics on 100 problems and the results showed that the RPW heuristic undoubtedly produced better results for the straight-line configuration [27]. Kim et al. used a modified version of RPW to find an initial solution and used a genetic algorithm to solve TALBP [28].

Simulation models help management by showing how post-change production will flow. Implementing a balanced line without applying simulation can cause millions of dollars in damage. Therefore, it is a must to see the results of the assembly line balancing by simulation model as many researchers did before in diverse industries [29–34].

Two-sided assembly lines consist of two connected serial lines in parallel. The products are worked concurrently at both sides of the line. This is generally the case when the product is mid-sized or larger. Studies mostly focus on one-sided ALB, with few studies targeting two-sided ALB. This type of ALBP

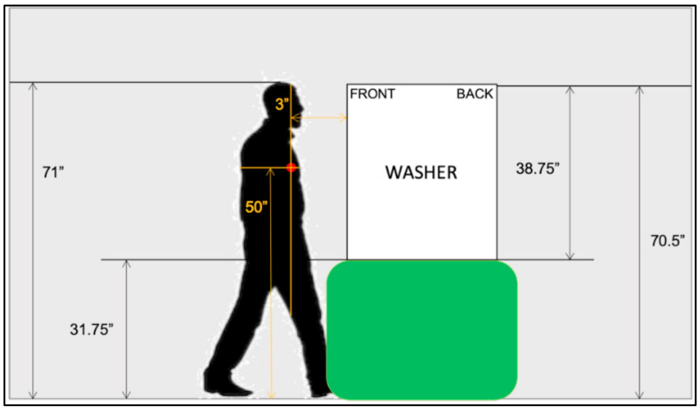

Figure 1. Distances used for MTM time estimation

was addressed and solved firstly by Bartholdi [35]. Baykasoglu and Dereli suggested the first study on the two-sided ALB with zoning constraints and they proposed an ant-colony-based heuristic algorithm to solve the problem [36]. A genetic algorithm was created by Kim et al. to solve two-sided ALB and they also presented a mathematical formulation to lead the future works on this area [37]. Gansterer and Hartl suggested SALBP and TALBP formulations with real-world constraints like task or machine incompatibilities and they provided feasible test instances for two-sided ALB [38].

Other characteristics of gALB have been considered recently. Pearce et al. presented an integer programming model and a combined heuristic of RPW, last-fit-increasing improvement and iterative blocking scheme for a complex two-sided ALB model of automotive production. They included several additional constraints for task-to-task relationships, work-station characteristics and parallel worker zoning interactions [39]. Chen et al. proposed a two-phased genetic algorithm and a mathematical formulation to solve ALBP including multi-skilled worker constraints [40]. Bautista et al. study the SALBP-1 with additional incompatibilities between tasks. They improved a greedy randomized adaptive search procedure and genetic algorithm to solve the problem [41].

3. Problem environment and supplementary constraints

The field of this study is a major household appliance industry and the washing machine is considered a representative product. Because it is a medium-sized product and appropriate for two people to work at the same time, it is produced on two-sided lines. There are some different types of characteristics listed below to ensure that the real-world system requirements are met. Additionally, the time estimation method we used is explained below.

3.1. Time Estimation

The individual task times are required to solve a real-world problem. Collecting task times with traditional times study methods requires so much time when the number of tasks is large. Thus, we used a validated technique for assembly time estimation called Methods Time Measurement (MTM). MTM is a time estimation method commonly used in both industry and academia [42,43]. MTM times were used for the estimation of common tasks like “get” and “place”. To ensure the time estimations are accurate as possible, a series of distances to key points on the machine were collected using the remaining body from the machine teardown. These distances were collected using a laser measure in several different machine orientations and station layout scenarios, as shown in Figure 1. Another advantage of using the MTM approach is that times can be adjusted based on the size and handling difficulty of parts, the body motions needed to perform assembly in hard-to-reach areas, and general fluctuations in assembly times. The remaining assembly tasks in the task model that could not be accurately estimated using MTM were collected using traditional time study techniques.

3.2. Zoning constraints

Zoning restrictions are introduced because the size of the work-piece may cause some restrictions. The washing machines are produced in two-sided lines means that parallel stations face the opposite side of the machine. Thus, we should consider different available working zones for sides of the line. A worker in a station cannot reach all the locations of the machine. For example, if a worker works on the front lower area of the machine, it is impossible to work on the back of the machine at the same time and also it is not ergonomic for the worker to work on the front upper side of the machine. Some stations have platforms to let workers work on the upper side of the machine. The zoning constraints were also used for ergonomic reasons to prevent excessive bending and reaching.

Each task is required to be done on a specific area of the machine. The product is divided into 16 pieces as shown in Figure 2 and the list of the product zones is defined as Z={DRBO, DRBI, DRFO, DRFI, DLBO, DLBI, DLFO, DLFI, URBO, URBI, URFO, URFI, ULBO, ULBI, ULFO, ULFI, URBT, ULBT, URFT, ULFT, URMT, ULMT}. Four letter product zone codes correspond to down (D) or up (U), right (R) or left (L), front (F) or back (B), inside (I) or outside (O) or top (T), respectively as shown in Figure 2. Tasks usually require access to more than one product zone and all these product

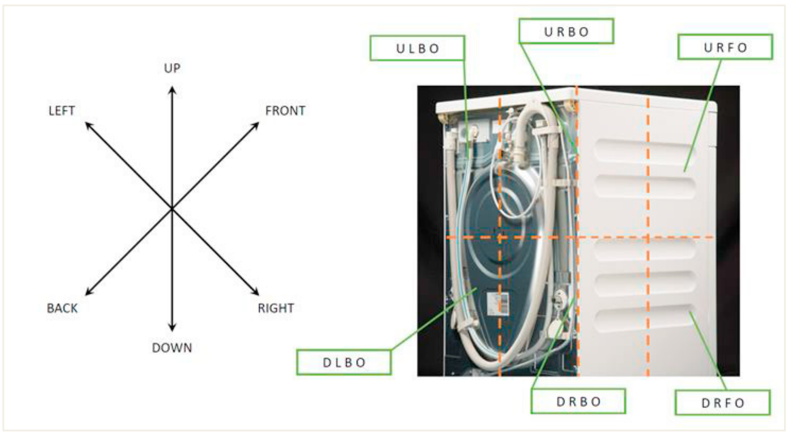

Figure 2. The product zone directions and work zones of machine

zones should be covered (accessible) by the station to assign the task to the station.

Each station is capable to reach different work zones depending on the configuration of the machine. Washing machines travel down the line in one of four configurations (front leading, back leading, right leading and left leading), which impacts where the workers can access the product. For example, if the machine is right leading, the front and back of the machine is accessible. Moreover, the presence of a platform may either hinder or facilitate access to certain product zones. The combination of machine orientation, platform and product zones are required to determine if a task can be assigned to a station. Work zones and their product zone coverages are given in Table 1. For example, if the machine is back-facing and there is no platform (NP) at a station, then a worker at that station can access the DRBO product zone.

Table 1: Work zone coverages

	Front		Back		Right		Left	
Product Zones	NP	P	NP	P	NP	P	NP	P
DRBO	0	0	1	0	1	0	0	0
DRBI	1	0	1	0	0	0	0	0
DRFO	1	0	0	0	1	0	0	0
DRFI	1	0	0	0	0	0	0	0
DLBO	0	0	1	0	0	0	1	0
DLBI	1	0	1	0	0	0	0	0
DLFO	1	0	0	0	0	0	1	0
DLFI	1	0	0	0	0	0	0	0
URBO	0	0	1	1	1	1	0	0
URBI	1	1	1	1	0	0	0	0
URFO	1	1	0	0	1	1	0	0
URFI	1	1	0	0	0	0	0	0
ULBO	0	0	1	1	0	0	1	1
ULBI	1	1	1	1	0	0	0	0
ULFO	1	1	0	0	0	0	1	1
ULFI	1	1	0	0	0	0	0	0
URBT	0	0	0	1	0	1	0	0
ULBT	0	0	0	1	0	0	0	1
URFT	0	1	0	0	0	1	0	0
ULFT	0	1	0	0	0	0	0	1
URMT	0	1	0	1	0	1	0	0
ULMT	0	1	0	1	0	0	0	1

3.3. Tooling and station dependent constraints

There are two types of tasks that require special tools or equipment. The first task set requires tools that are available at more than one station. Thus, tool required tasks are not fixed to the station with tools. The second task set is the fixed tasks which require special equipment that is available at only one station. Thus, fixed tasks are pre-assigned to the fixed stations. These tool and equipment requirements are added to the model as set of constraints in addition to the precedence and zoning constraints.

3.4. Adjacency Constraints

Some tasks are required to be processed consecutively by the same worker. These tasks are called adjacent tasks. Consider that task A is to get an object and task B is to install that object to the washing machine. In that case, task B needs to be done right after task A since the worker's hands will be unavailable to accomplish any other tasks. These tasks are modelled separately because their standard times differ based on aspects independently, such as part location on the side of the line.

4. Integer programming

We present integer programming formulation for both type-1 and type-2 gALBs to meet the demand of our industry partner. In our specific problem, it is observed that most of the tasks are required to be assigned to the front face of the machine. So, the probability of a task and its immediate predecessor task to be assigned to a mated station is low. Thus, the problem is formulated as a one-sided gALB problem for the integer programming and precedence check for the mated stations is done as a post-processing step. These formulations are extensions of formulations such those proposed by Pearce et al. [39] but include exactly the characteristics we want. As exact models, they serve as our benchmark for the metaheuristic.

The constraints mentioned in section 3 are included in the formulation. The sets and input parameters used in the formulation are given in Table 2 and Table 3, respectively.

Table 2: Sets

Symbol	Description	Index
I	Set of tasks	i
K	Set of stations	k
F	Set of fixed stations	k
N	Set of non-machinery fixed stations	k
M	Set of machinery fixed stations	k
T_k	Set of tasks fixed at station k, $\forall k \in F$	i
P_i	Set of all predecessors of task i	i
S_i	Set of all successors of task i	i

Table 3: Input Parameters

Symbol	Description	
C	Cycle time (sec)	Note: Parameter for only model type-1
W	Number of work locations	Note: Parameter for only model type-2
t_i	Duration of task i	
E_i	Earliest station that task i can be assigned	
L_i	Latest station that task i can be assigned	
τ_i^P	Total duration of tasks in P_i	
τ_i^S	Total duration of tasks in S_i	
τ_k	Total duration of fixed tasks assigned to station k, $\forall k \in F$	
p_{ij}	$\begin{cases} 1, \text{if task } i \text{ is an immediate predecessor of task } j \\ 0, \text{otherwise} \end{cases}$	
a_{ij}	$\begin{cases} 1, \text{if tasks } i \text{ and } j \text{ are adjacency tasks} \\ 0, \text{otherwise} \end{cases}$	
q_{ik}	$\begin{cases} 1, \text{if task } i \text{ has to be assigned to station } k \\ 0, \text{otherwise} \end{cases}$	
s_{ik}	$\begin{cases} 1, \text{if task } i \text{ can be assigned to station } k \text{ due to work zone accessibility} \\ 0, \text{otherwise} \end{cases}$	
e_{ik}	$\begin{cases} 1, \text{if task } i \text{ is able to assigned to station } k \text{ due to equipment necessity} \\ 0, \text{otherwise} \end{cases}$	
r_{ik}	$\begin{cases} 1, \text{if } s_{ik}, e_{ik}\ equal\ 1\ and\ fixed\ station\ constraints\ allow \\ 0, \text{otherwise} \end{cases}$	

4.1. Decision variables

The decision variables for both models are given in Table 4. Note that c_k and c_{max} are variables for the only type-2 model, while x_{ik} and y_k are for both models. To eliminate unnecessary decision variables x_{ik} is defined only if r_{ik} equals one. The factors that affect the formation of r_{ik} are explained in the pre-processing section.

Table 4: Decision Variables

Symbol	Description	Valid for type
x_{ik}	$\begin{cases} 1, \text{if task } i \text{ is assigned to station } k \\ 0, \text{else} \end{cases}$	I and II
y_k	$\begin{cases} 1, \text{if station } k \text{ is active} \\ 0, \text{else} \end{cases}$	I and II
c_k	Cycle time of station k	Only II
c_{max}	$max\{c_k \mid k \in K\}$	Only II

4.2. Pre-processing

Commercial solvers (e.g. GUROBI, CPLEX) have improved drastically in recent years with the development of advanced integer programming solution methods. Depending on the size and nature of the problem, these commercial solvers are even able to solve NP-hard problems in a reasonable time. We reduced the number of variables in two ways; by computing the earliest and latest available stations for each task [44] and using environment restrictions.

The earliest and latest stations of tasks are computed as follows:

$$E_i = \max\left\{(\max\{k\} \mid \forall i \in P_i, \forall k \in F, q_{ik} = 1), \left\lceil \tau_i^P / C \right\rceil\right\} \quad (1)$$

$$L_i = \min\left\{\min\{k\} \mid \forall i \in S_i, \forall k \in F, q_{ik} = 1, \left\lceil |K| + 1 - \tau_i^S / C \right\rceil\right\} \quad (2)$$

To find the earliest station of task i, τ_i^P is divided by the cycle time and rounded up. The resulting number is compared with the station number k which is the largest station ID among the stations that any fixed task in P_i is fixed to. E_i equals to the maximum of these two numbers. To find the latest station of task i, τ_i^S is divided by the cycle time and rounded up, then it is subtracted from the total number of stations plus one. The resulting number is compared with the station number k which is the smallest station ID among the stations that any fixed task in P_i is fixed to. L_i equals to the minimum of these two numbers.

A parameter matrix r_{ik} is used to decide which variables to define. We can think of two groups of tasks: fixed tasks, and unfixed tasks. If task i is a fixed task and in set T_k, r_{ik} equals zero for all stations except the station k. If task i is an unfixed task, then r_{ik} equals one when $s_{ik}, e_{ik} = 1$ and $E_i \le k \le L_i$. There is a special case for fixed stations. If station k is in the set M or k is in the set N and $\tau_k < C$, then r_{ik} equals zero for all the tasks except the tasks in T_k. This pre-processing method allows us to eliminate the fixed resource constraints, work-zone constraints, and station interval constraints. As the variable x_{ik} is only defined when r_{ik} equals one, this condition is not written for each constraint that includes variable x.

4.3. Formulation

Objective functions and distinctive constraints of type-1 and type-2 gALB are given in Table 5 and Table 6, respectively. Subsequently, the common constraints for the models are given in Table 7. Types of constraints and objectives are grouped into three categories. “T1” is only for the model type-1, “T2” is only in the model type-2, and “C” constraints are the common constraints involved in both models.

Table 5: Mathematical Model of Type-1 gALB

Constraint Formula	Quantification and Condition	Set
$Min \sum_{k \in K} y_k$		(T1)
$\sum_{i \in I}(t_i x_{ik}) \le C y_k$	$\forall k \in K, \forall k \notin F\ or\ \forall k \in F, \tau_k < C$	(T1.1)

The objective of the type-1 model (T1) is to minimize the total number of active work locations with given cycle time. Constraints (T1.1) imply that the workload assigned to each work station cannot exceed the cycle time.

Table 6: Mathematical Model of Type-2 gALB

Constraint Formula	Quantification and Condition	Set
$Min\ c_{max}$		(T2)
$\sum_{k\in K} y_k \le W$		(T2.1)
$\sum_{i\in I}(t_i x_{ik}) = c_k$	$\forall k \in K$	(T2.2)
$c_k \le c_{max}$	$\forall k \in K$	(T2.3)
$\sum_{i\in I}(t_i x_{ik}) \le N y_k$	$\forall k \in K$	(T2.4)

The objective of the type-2 model (T2) is to minimize the cycle time with a given total available number of work locations. Constraints (T2.1) guarantee that the number of active work-stations is less than or equal to the number of available work locations. Constraints (T2.2) impose that the cycle time of a workstation equals the total duration of the tasks assigned to that station. Constraint set (T2.3) helps the c_{max} variable to be larger than or equal to the longest cycle time. Constraints (T2.4) ensure that a station with an assigned task becomes active.

Table 7: Common Constraints for Type-1 and Type-2 Models

Constraint Formula	Quantification and Condition	Set
$\sum_{k\in K} x_{ik} = 1$	$\forall i \in I$	(C1)
$\sum_{k=v+1}^{\lvert K\rvert} x_{ik} \le 1 - \sum_{k=1}^{v} x_{jk}$	$\forall i,j \in I,\ \forall v = 1 \ldots \lvert K\rvert - 1, if\ p_{ij} = 1$	(C2)
$x_{ik} = x_{jk}$	$\forall i,j \in I,\ \forall k \in K,\ if\ a_{ij} = 1$	(C3)
$x_{ik} \in \{0,1\}$	$\forall i \in I,\ \forall k \in K$	(C4)
$y_k \in \{0,1\}$	$\forall k \in K$	(C5)

Constraint set (C1) ensures that all regular tasks are assigned to exactly one work station. Set (C2) enforces precedence constraints. Constraint set (C3) guarantees that the adjacency tasks are assigned to the same stations. Some tasks require fixed resources in exact stations since it is costly to change the location of some equipment and machines. Constraints (C4) and (C5) ensure that all decision variables are binary.

4.4. Post-processing

The problem is formulated in a way that the order of the tasks inside stations is not addressed. Once the tasks are assigned to stations, we run an algorithm to order the tasks inside the station according to precedence and adjacency constraints for each station. Likewise, the precedence relationship inside the mated stations is not included in the formulation. A post-process is used to check this issue. In the case of mated station precedence violation, extra constraints to block this violation is added and run the integer program again. Consider a result in which we solved the problem and task A is the immediate predecessor of task B and they are at the same mated group. If the cycle time and the orders of tasks inside mated stations do not allow task B to be done after task A, then there is a violation. We add a constraint not to allow task A and task B to be assigned these mated stations at the same time. In this way, we keep the integer programming as small as possible.

5. Metaheuristic

Metaheuristics are often used to solve large assembly line balancing problems because they can solve problems more quickly than integer programs even though they do not guarantee an optimal solution. This was developed to minimize the number of stations required for the production line, and the constraints from the integer program were incorporated into the metaheuristic.

5.1. Initial solution

To use this metaheuristic, first a feasible line balance needs to be created. For this project, the ranked positional weight technique was used [26]. First, it creates an order for the tasks to be assigned in by using precedence and task time to weight how early the tasks should be placed, $1 + (\tau_i^P/C)$ gives the earliest station due to task times. Because precedence is preserved, the balance is always feasible. In order of their ranked positional weight, tasks are placed in stations where precedence, station attributes, and cycle time allow. After all the tasks have been allocated, a count of how many stations were used is obtained. Algorithm 1 is a pseudocode to create the initial solution.

Algorithm 1: Assignment order using RPW

```
 1: function RPW
 2:     assign fixed tasks to appropriate stations
 3:     for each task in task list do
 4:         for each station in model do
 5:             if all requirements are met then
 6:                 place task in selected station
 7:             end if
 8:         next
 9:     next
10: end function
```

5.2. Hill climbing algorithm

Using the initial solution, the hill climbing algorithm selects the station that has the smallest allotted time. It attempts to move each task to different feasible stations and closes stations with no tasks in them. If the station cannot be closed, it is added to the tabu list, the remaining station with the shortest allotted time is selected, and the process is repeated. Pseudocode is given in Algorithm 2.

Algorithm 2: Hill Climbing with Tabu List

```
 1: function HillClimbing
 2:     while fewer than 10 failed attempts do
 3:         select station with shortest time
 4:         for each task assigned to the station do
 5:             for each station in use do
 6:                 if all requirements are met then
 7:                     move task to new station
 8:                 end if
 9:             next
10:         next
11:         if all tasks in shortest station are moved then
12:             close station
13:         else
14:             add shortest station to tabu list to prevent looping
15:         end if
16:         if no tasks were moved then
17:             add an additional failed attempt
18:         end if
19:     loop
20: end function
```

5.3. Abandoned hill climbing algorithm with tabu list

From this baseline, the hill climbing algorithm starts, and switches the position of tasks in the order. If the resulting order will satisfy precedence, it will run through the check to allocate tasks. If the new balance is feasible and requires fewer stations than the previous, it is adopted as the new balance. By checking multiple swaps, many possible solutions were sampled. The algorithm was abandoned because little improvement was seen between the initial solution and the final solution. It is believed that this lack of improvement was because the line balance was similar even when tasks were swapped. Algorithm 3 is a pseudocode to create the initial solution.

Algorithm 3: Abandoned Hill Climbing

```
1:  function AbandonedHillClimbing
2:    for each task in list do
3:      for each task after previous task do
4:        switch the previous two tasks
5:        test for precedence
6:        if precedence can be met then
7:          using new order create new balance
8:          if fewer stations are needed then
9:            replace original balance
10:         end if
11:       end if
12:     next
13:   next
14: end function
```

6. Simulation modeling

A simulation model is developed in Arena (a commercial simulation software product) to (1) understand and represent the current-state processes, (2) track metrics that help identify inefficiencies in the system, and (3) provide a validation tool for testing various throughput improvement scenarios developed by the research team and the project sponsors.

6.1. Methodology

The simulation project follows a standard template involving problem formulation, identifying objectives, collecting input data, developing a base model, verifying the processes, validating the current-state operations, developing pertinent metrics to measure the objectives, developing scenarios to be tested, conducting simulation runs, collecting and analyzing output, and documenting results.

6.1.1. Objective

The objective is to validate the expected throughput increase of the assembly line after incorporating the results from the line-balancing exercise. The output of the line-balancing exercise is the optimal number and cycle times of tasks at each workstation, which serves as the input to the simulation model.

6.1.2. Inputs required

The primary inputs require, which are the number of tasks at each workstation and the process time for each task, are organized in a spreadsheet in Excel. Flexibility in changing various input parameters – add new tasks at a workstation, edit task data, remove a task from a workstation, or move a task within or between workstations – is built into the input file using VB macros.

6.1.3. Developing base model and metrics

Initially, a base model is developed along with relevant metrics to validate the current state of operations for the machining section of one of the assembly lines. These metrics includes total units produced in a shift, hourly throughput, and station utilization. Once the processes in the machining section are validated, the model is used as the standard for building the remaining sections – assembly, inspection, and final.

6.1.4. Incorporating variability in task times

After the initial model is created, an option to conduct simulations with or without variability in task times is incorporated, resulting in two scenarios:

- Ideal task-times scenario – The workstations in the model used the most likely time specified for each task, thus eliminating variability due to process times.
- Stochastic task-times scenario – The workstations in the model used a triangular distribution of the times for each task (i.e., a minimum time, a most likely time, and a maximum time) in order to account for actual variability in processing times for workstations where an operator is assigned to complete tasks. Note that the workstations with machines are expected to have far less variability than the workstations where operators performs manual tasks.

6.1.5. Incorporating loss in production time

After analyzing data for loss in production time recorded over the past six months, the following reasons are identified and incorporated into the model:

- Machine breakdowns
- Internal material shortages and material defects
- Process unbalance

The model simulates lost production time in the form of mean time between failures (MTBF) and the mean time to repair (MTTR), which are calculated based on the received input data. The MTBF is computed by dividing the total scheduled production time by the total number of breakdowns for each reason for loss. The MTTR is computed by dividing the total downtime by reason for loss by the total number of breakdowns for each reason for loss. Both the MTBF and MTTR are exponentially distributed. Further, since the analyzed data only captured lost time if it is greater than five minutes, an additional five minutes is added to the MTTR.

6.2. Simulation scenarios based on downtime

Simulations can be conducted with or without incorporating downtimes, with further flexibility in choosing any combination of reasons for loss in production time. One

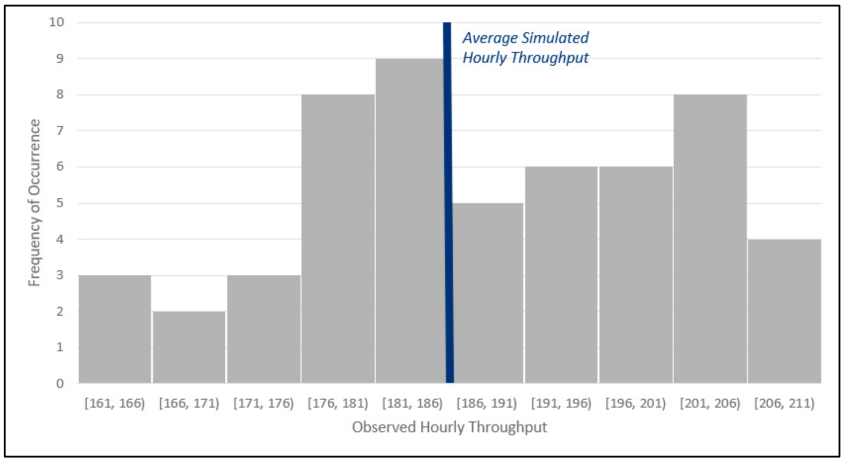

Figure 3. Comparison of observed and simulated hourly throughput for data from May to August 2019

example would be to test only the effect of machine breakdowns on the throughput of the lines. The locations on the line that cause a loss in time are identified from meetings with subject matter experts at the plant. It is determined that process unbalance is most likely to occur at one of two specific locations, while material defect and internal material shortage is most likely to occur at six other locations. The machines that are most likely to breakdown are not known with certainty, so three machines in different sections of the line are selected initially. At first, the likelihood that a downtime would occur is equally distributed between these locations. After further discussions with the subject matter experts, an option to test the effect of downtime occurring at a specific machine on the throughput of the line is incorporated. Functionality that allows the user to alter the percentage of time a breakdown occurred at any given machine is also added.

6.3. Model validation

The above scenarios are simulated with ideal and stochastic task times for five replications of 20 shifts each with five-hundred and eighty minutes of available production time. The results are collected after a warm-up period of three hours, introduced to achieve stable running conditions. Output statistics, including total units produced in a shift, hourly throughput, and station utilization, are collected for each of the scenarios. For validating the model, the results are compared with the actual production data collected over the same time period. The model produces comparable production levels over the time period from May to August 2019. Figure 3 reports the observed hourly throughput for the 100 simulated shifts and the average simulated hourly throughput.

7. Numerical Results

Two integer programs, type-1 and type-2 objective, and a metaheuristic have been developed to offer various solutions to our industrial partner. Although the integer program provides better results than the metaheuristic in terms of the number of stations used, it requires a commercial solver to be bought and has a longer running time. The results are presented by comparing the integer program and metaheuristic to show the application advantages and disadvantages of the methods.

There are 315 individual tasks and 175 available work stations in our problem. The number of available work stations is more than twice the number of currently active stations - 73 stations are active presently.

7.1. Integer program and metaheuristic results

The integer program is modeled in Python 3.7.1 and solved through GUROBI 7.0.1 solver. Finding the optimal solution for the type-1 integer programming model takes around ten minutes on a 64 bit Intel Core i7-8550U CPU (1.80 GHz) with 32 GB of RAM computer. The runtime and the optimality gap for each type-2 integer program scenario are given in Table 8. Although Python is an open-source programming language, GUROBI is not free for non-academic purposes.

The metaheuristic algorithm is coded and solved in Visual Basic for Applications (VBA). The runtime to find the best solution takes approximately 1.5 seconds on a 64 bit Intel Core i7-8565U CPU with 16 GB of RAM computer. VBA is not an open-source programming language. However, most companies have access to the Microsoft Office including our partner. Therefore, there is no need to purchase a software to run the ALB metaheuristic regularly. The metaheuristic could have been written in another open-source programming language other than VBA, however, our partner has already Microsoft Office access and it is a plus that the engineers who will implement the balanced assembly line are familiar with VBA.

Focusing on the type-1 objective, the integer program has found the optimal solution as 59 work stations and the best objective found by the metaheuristic is 67 work stations. While the integer program decreases the number of required stations by nearly 19%, metaheuristic decreases around 8%, compared to the current implemented solution. In companies where the rate of the absentee is low, a 19% decrease in the need for workers can provide large long-term profit. It should be noted that the integer program may not find the optimal solution for

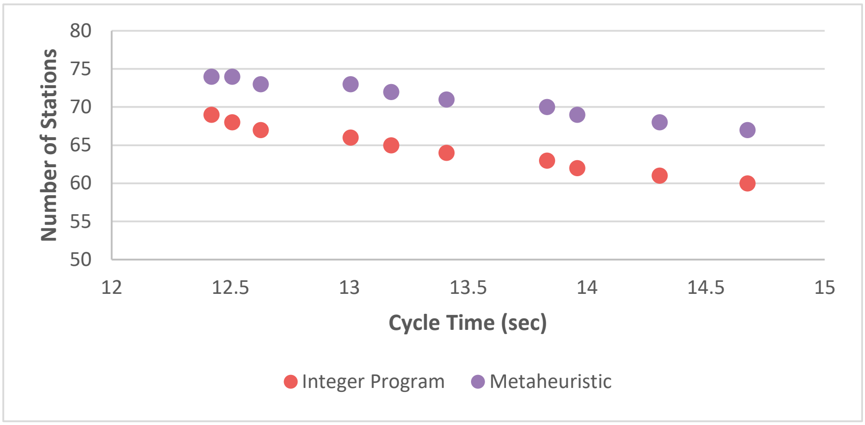

Figure 4. Comparison of IP and metaheuristic results

larger problems. The integer program will be able to provide a worthy solution unless the problem size increases drastically. The metaheuristics are more reliable in terms of generating feasible solutions for larger problems, since the integer program may run of out memory before a feasible solution is found.

Focusing on the type-2 objective, we executed the integer program with a given maximum number of stations ranging from 59 to 69, resulting in the minimum cycle time for each scenario; the integer program was given 30 minutes of solving time, resulting in optimality gaps. After solutions were gathered from the type-2 integer program, each cycle time was used as a parameter for the metaheuristic. Formally, the results of the metaheuristic were compared with the input parameters of the integer program in Figure 4. Furthermore, for better understanding of the optimality of the type-2 integer program and the runtime comparison with metaheuristic is summarized in Table 8.

Table 8: Comparison between Integer Program and Metaheuristic

Cycle Time	Gap in IP	IP Run Time (s)	MH Run Time (s)
12.42	0.00%	102.74	1.9
12.50	0.70%	1810.28	1.75
12.62	1.64%	1811.98	1.77
13.00	4.50%	1811.81	1.33
13.17	4.89%	1811.03	1.36
13.40	5.73%	1811.13	1.75
13.83	7.88%	1810.62	1.35
13.95	7.17%	1811.23	1.81
14.30	8.57%	1811.34	1.64
14.67	9.79%	1810.74	1.35

7.2. Simulation results

Results were gathered for six scenarios:

- Current-state – with and without incorporating downtime
- Balanced line – with and without incorporating downtime
- Balanced line with air-driver adjustments – with and without incorporating downtime

Note:

- The balanced line refers to task allocations at workstations obtained using the integer program and the metaheuristic.
- The task times for all scenarios utilized the MTM times.
- "Air-driver adjustments" imply that air-driver tools could be installed at any of the workstations on the line, thus eliminating some of the restrictions on task allocations resulting in cycle time reduction.

Table 9 summarizes these results along with expected improvement. The hourly throughput results achieved in simulations using the integer program and the metaheuristic were the same despite different task allocations because both methods were constrained by the same bottleneck workstation that drove throughput. In other words, a single work station task allocations set the cycle time for the entire line and is the slowest station, and results in being the bottleneck.

Table 9: Simulation Results

Scenarios	**Simulated Hourly Throughput**		**Throughput Improvement**	
	No Downtime	***With Downtime***	***Per hour***	***Per shift***
Current-state	178	168	-	-
Balanced line	211	196	28	215
Balanced line with air-driver adjustments	237	216	48	368

8. Conclusion

This paper is conducted to balance a major household appliance assembly line and washing machine is selected as representative product. We balance a TALBP with additional environment constraints in two different ways: integer program and metaheuristic. The real world environment requirements for the washing machine assembly line are introduced. The mathematical formulation for IP and the algorithm for hill climbing with tabu list are proposed. Since the task distribution allows, integer program is modeled as one-sided assembly line balancing problem. In this way, we are able to solve the problem in optimal way with post-processing to ensure that the results generally do not have mated station precedence violations. The main contribution of this paper is solving a case study with the real-world environment requirements by comparing integer programming and a metaheuristics with the detailed pros and cons of these two methods. Additionally, the size of the integer programming formulation is reduced by applying some problem specific techniques to be able to solve the problem in a reasonable time which can be very helpful for the process of solving other case studies.

The results obtained from both methods are better than the current system consequently both IP and metaheuristics can be used for the long-term planning. The pros and cons of the two methods are discussed considering the numerical results, time requirement, software requirement and the usability. Furthermore, the simulation model is used to examine the flow of the balanced line for different scenarios. The model is executed for both IP and metaheuristic results. Since the same cycle time used for both models, the bottleneck stations have the same workload. Therefore, the hourly throughput for the results gathered from different methods have the same number of products. Nevertheless, the hourly throughput is increased significantly compared to current production especially when the line is flexible about the air-driver adjustments. Consequently, our results show that the company has an inefficient assembly line with respects to number of workers, workload balance, number of stations, and hourly throughput. The company can adapt to the proposed assembly line by replacing some tools and educating workers for the new set of tasks in a couple of weeks.

Acknowledgements

This project was funded by Samsung Electronics Home Appliances.

References

[1] Bryton B. Balancing of a Continuous Production Line. Northwestern University, Evanson, ILL, 1954.

[2] Karp RM. Reducibility Among Combinatorial Problems. Springer, Boston, MA; 1972.

[3] Salveson ME. The Assembly Line Balancing Problem. J Ind Eng 1955;6:18–25.

[4] Baybars İ. A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem Author (s): İlker Baybars Published by : INFORMS Stable URL : http://www.jstor.org/stable/2631657. Manage Sci 1986;32:909–32.

[5] Scholl A. Balancing and sequencing of assembly lines. 2nd ed. Heidelberg ; New York : Physica-Verlag; 1999.

[6] Sivasankaran P, Shahabudeen P. Literature review of assembly line balancing problems. Int J Adv Manuf Technol 2014;73:1665–94. https://doi.org/10.1007/s00170-014-5944-y.

[7] Wee TS, Magazine MJ. Assembly line balancing as generalized bin packing. Oper Res Lett 1982;1:56–8.

[8] Bowman EH. Assembly Line Balancing by Linear Programming. Oper Res 1960;8:385–9. https://doi.org/10.11398/economics1950.5.3-4_156.

[9] White WW. Letter to the Editor—Comments on a Paper by Bowman. Oper Res 1961;9:274–6. https://doi.org/10.1287/opre.9.2.274.

[10] Thangavelu SR, Shetty CM. Assembly Line Balancing by Zero-One Integer Programming. AIIE Trans 1971;3:61–8.

[11] Salama S, Abdelhalim A, Eltawil AB. Mathematical modeling approaches to solve the line balancing problem. ICORES 2017 - Proc 6th Int Conf Oper Res Enterp Syst 2017;2017-Janua:401–8. https://doi.org/10.5220/0006199404010408.

[12] Scholl A, Becker C. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. Eur J Oper Res 2006;168:666–93. https://doi.org/10.1016/j.ejor.2004.07.022.

[13] Cerqueus A, Delorme X. A branch-and-bound method for the bi-objective simple line assembly balancing problem. Int J Prod Res 2018;57:5640–59. https://doi.org/10.1080/00207543.2018.1539266.

[14] Scholl A, Klein R. 1997_B&B_Salome_SALBP.pdf. INFORMS J Comput 1997;9:319–35.

[15] Klein R, Scholl A. Maximizing the production rate in simple assembly line balancing - A branch and bound procedure. Eur J Oper Res 1996;91:367–85. https://doi.org/10.1016/0377-2217(95)00047-X.

[16] Johnson R V. Branch and Bound Algorithm for Assembly Line Balancing Problems With Formulation Irregularities. Manage Sci 1983;29:1309–24. https://doi.org/10.1287/mnsc.29.11.1309.

[17] Li Z, Kucukkoc I, Zhang Z. Branch, bound and remember algorithm for U-shaped assembly line balancing problem. Comput Ind Eng 2018;124:24–35. https://doi.org/10.1016/j.cie.2018.06.037.

[18] Azizoğlu M, İmat S. Workload smoothing in simple assembly line balancing. Comput Oper Res 2018;89:51–7. https://doi.org/10.1016/j.cor.2017.08.006.

[19] Hoffmann TR. Eureka. A hybrid system for assembly line balancing. Manage Sci 1992;38:39–47. https://doi.org/10.1287/mnsc.38.1.39.

[20] Wu EF, Jin Y, Bao JS, Hu XF. A branch-and-bound algorithm for two-sided assembly line balancing. Int J Adv Manuf Technol 2008;39:1009–15. https://doi.org/10.1007/s00170-007-1286-3.

[21] Held M, Karp RM, Shareshian R. Assembly-Line Balancing—Dynamic Programming with Precedence Constraints. Oper Res 1963;11:442–59. https://doi.org/10.1287/opre.11.3.442.

[22] Bautista J, Pereira J. A dynamic programming based heuristic for the assembly line balancing problem. Eur J Oper Res 2009;194:787–94. https://doi.org/10.1016/j.ejor.2008.01.016.

[23] Kao EPC. Preference Order Dynamic Program for Stochastic Assembly Line Balancing. Manage Sci 1976;22:1079–104. https://doi.org/10.1287/mnsc.22.10.1097.

[24] Leitold D, Vathy-Fogarassy A, Abonyi J. Empirical working time distribution-based line balancing with integrated simulated annealing and dynamic programming. Cent Eur J Oper Res 2019;27:455–73. https://doi.org/10.1007/s10100-018-0570-7.

[25] Li Y, Coit D. Priority rules-based algorithmic design on two-sided assembly line balancing. Prod Eng 2018;12:95–108. https://doi.org/10.1007/s11740-017-0786-8.

[26] Helgeson WB, Birnie DP. Assembly line balancing using the ranked positional weight technique. J Ind Eng 1961;12:394–8.

[27] Fathi M, Fontes DBMM, Urenda Moris M, Ghobakhloo M. Assembly line balancing problem: A comparative evaluation of heuristics and a computational assessment of objectives. J Model Manag 2018;13:455–74. https://doi.org/10.1108/JM2-03-2017-0027.

[28] Kim YK, Kim Y, Kim YJ. Two-sided assembly line balancing: A genetic algorithm approach. Prod Plan Control 2000;11:44–53. https://doi.org/10.1080/095372800232478.

[29] Driscoll J, Abel Shafi AA. A simulation approach to evaluating assembly line balancing solutions. Int J Prod Res 1985;23:975–85. https://doi.org/10.1080/00207548508904760.

[30] Das B, Sanchez-Rivas JM, Garcia-Diaz A, MacDonald CA. A computer simulation approach to evaluating assembly line balancing with variable operation times. J Manuf Technol Manag 2010;21:872–87. https://doi.org/10.1108/17410381011077964.

[31] Büyüksaatçi S, Tüysüz F, Bilen K. Balancing and simulation of assembly line in an LCD manufacturing company. 6th Int Conf Model Simulation, Appl Optim ICMSAO 2015 - Dedic to Mem Late Ibrahim El-Sadek 2015:1–5. https://doi.org/10.1109/ICMSAO.2015.7152254.

[32] Mendes AR, Ramos AL, Simaria AS, Vilarinho PM. Combining heuristic procedures and simulation models for balancing a PC camera assembly line. Comput Ind Eng 2005;49:413–31. https://doi.org/10.1016/j.cie.2005.07.003.

[33] Kitaw D, Matebu A, Tadesse S. Assembly Line Balancing Using Simulation Technique. J EEA 2010;27:69–80.

[34] Jamil M, Razali NM. Simulation of Assembly Line Balancing in Automotive Component Manufacturing. IOP Conf Ser Mater Sci Eng 2016;114. https://doi.org/10.1088/1757-899X/114/1/012049.

[35] Bartholdi JJ. Balancing Two-Sided Assembly Lines: A Case Study. Int J Prod Res 1993;31:2447–61.

[36] Baykasoglu A, Dereli T. Two-sided assembly line balancing using an ant-colony-based heuristic. Int J Adv Manuf Technol 2008;36:582–8. https://doi.org/10.1007/s00170-006-0861-3.

[37] Kim YK, Song WS, Kim JH. A mathematical model and a genetic algorithm for two-sided assembly line balancing. Comput Oper Res 2009;36:853–65. https://doi.org/10.1016/j.cor.2007.11.003.

[38] Gansterer M, Hartl RF. One-and two-sided assembly line balancing problems with real-world constraints. Int J Prod Res 2018;56:3025–42. https://doi.org/10.1080/00207543.2017.1394599.

[39] Pearce BW, Antani K, Mears L, Funk K, Mayorga ME, Kurz ME. An effective integer program for a general assembly line balancing problem with parallel workers and additional assignment restrictions. J Manuf Syst 2019;50:180–92. https://doi.org/10.1016/j.jmsy.2018.12.011.

[40] Chen JC, Chen YY, Chen TL, Kuo YH. Applying two-phase adaptive genetic algorithm to solve multi-model assembly line balancing problems in TFT–LCD module process. J Manuf Syst 2019;52:86–99. https://doi.org/10.1016/j.jmsy.2019.05.009.

[41] Bautista J, Suarez R, Mateo M, Companys R. Local search heuristics for the assembly line balancing problem with incompatibilities between tasks. Proc - IEEE Int Conf Robot Autom 2000;3:2404–9. https://doi.org/10.1109/robot.2000.846387.

[42] Ferreira JCE. Analysis of the Methods Time Measurement (MTM) Methodology through its Application in Manufacturing Companies Analysis of the Methods Time Measurement (MTM) Methodology through its Application in Manufacturing Companies. Flex Autom Intell Manuf 2015;1:9. https://doi.org/10.13140/RG.2.1.2826.1927.

[43] Renu RS, Mocko G, Koneru A. Use of big data and knowledge discovery to create data backbones for decision support systems. Procedia Comput Sci 2013;20:446–53.

[44] Scholl A, Fliedner M, Boysen N. Absalom: Balancing assembly lines with assignment restrictions. Eur J Oper Res 2010;200:688–701. https://doi.org/10.1016/j.ejor.2009.01.049.