European Journal of Operational Research 229 (2013) 106–113

Contents lists available at SciVerse ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Production, Manufacturing and Logistics

An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time

Mariona Vilà, Jordi Pereira *

Escola Universitària d'Enginyeria Tècnica Industrial de Barcelona (EUETIB), Universitat Politècnica de Catalunya, C. Comte Urgell, 187 1st Floor, 08036 Barcelona, Spain

ARTICLE INFO

Article history:
Received 2 February 2012
Accepted 1 March 2013
Available online 14 March 2013

Keywords:
Branch and bound
Manufacturing
Assembly line balancing

ABSTRACT

In this article, we present a new exact algorithm for solving the simple assembly line balancing problem given a determined cycle time (SALBP-1). The algorithm is a station-oriented bidirectional branch-and-bound procedure based on a new enumeration strategy that explores the feasible solutions tree in a non-decreasing idle time order. The procedure uses several well-known lower bounds, dominance rules and a new logical test based on the assimilation of the feasibility problem for a given cycle time and number of stations (SALBP-F) to a maximum-flow problem.

The algorithm has been tested with a series of computational experiments on a well-known set of problem instances. The tests show that the proposed algorithm outperforms the best existing exact algorithm for solving SALBP-1, verifying optimality for 264 of the 269 benchmark instances.

1. Introduction

An assembly line is a production system most commonly used in the flow-line production of goods on an industrial scale. It consists of m workstations connected by a conveyor belt or similar mechanical device. As unfinished products travel from station to station, certain operations are performed to obtain the final product. The indivisible operations necessary to finish a product are known as tasks $j = 1,\ldots,n$ and require a fixed operation time t_j to be completed.

The conveyor belt speed determines the output rate and the time in which each task must be completed before moving on to the next station. The available time is the same for every station and is referred to as cycle time c. The total operation time of the tasks assigned to each station can never exceed the cycle time.

Additionally, some tasks may not be able to start until another task has been completed. These precedence constraints must be fulfilled (i.e., if any task i assigned to station k precedes a task j assigned to station h, then $h \geqslant k$ must hold).

The simple assembly line balancing problem type 1 (SALBP-1) consists of assigning each task to a station and fulfilling the precedence constraints without exceeding the given cycle time c. The goal of this optimisation problem is to maximise the line efficiency by minimising the number of stations used m. This process is equivalent to minimising the total idle time for all stations.

Note that SALBP-1 is a reversible problem. As such, the precedence constraints may be reversed to obtain a different instance, and any solution found for the new instance can be converted to a solution for the original instance. Tasks can also be assigned to workstations in reverse order, and the initial or final workstations can even be constructed simultaneously.

Before describing the proposed algorithm in further detail, we present a quick review of the existing literature concerning SALBP-1.

Over the last 60 years [8], numerous procedures have been proposed for solving this problem. A detailed review is available in [15]. These procedures may be classified into three groups:

- Constructive (or *greedy*) algorithms, based on assigning tasks to stations using a priority rule. These procedures rarely yield an optimal solution for large instances, but they can offer sufficient solutions for practical purposes in a short computation time [19].
- Enumerative algorithms, which have been proven to yield the best results for SALBP-1. Enumerative algorithms may be classified as truncated, graph-based and tree-search-based procedures. The truncated algorithms are heuristic procedures [5,9], while graph-based algorithms are usually dynamic programming approaches [2,7,11]. Among tree-search-based procedures, branch-and-bound procedures perform particularly well [10,12,16].
- Metaheuristic procedures, which have the advantage of being easily applicable to general problems, although their results are not usually as good as those obtained by exact procedures. This group includes Tabu Search procedures [17], ant colony optimisation [1] and genetic algorithms [6].

* Corresponding author. Tel.: +34 934011096; fax: +34 934016054.
E-mail addresses: mariona.vila.bonilla@upc.edu (M. Vilà), jorge.pereira@upc.edu (J. Pereira).

0377-2217/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.ejor.2013.03.003

Our proposed algorithm belongs to the second group: enumerative algorithms. It is a station-oriented, bidirectional branch-and bound procedure with a new ramification strategy, which enumerates feasible task assignments to stations – identified as the nodes of the branch-and-bound tree – in a non-decreasing idle time order. This branching method allows the use of a strengthened variation of Johnson's dominance rule [12]. The algorithm also includes Jackson's dominance rule [11] and the Schrage–Baker labelling dominance rule [18]. The bounding strategy is based on a well-known set of seven lower bounds [14] and a new logical test based on the well-known maximum flow problem. Additionally, several preprocessing rules are applied to the problem to increase the efficiency of the algorithm.

The remainder of this manuscript is organised as follows. Section 2 details the preprocessing of the problem. Section 2.1 focuses on the lower bounds, reduction techniques and logical tests used in the algorithm, while Section 2.2 describes a *greedy* heuristic and reviews the well-known Hoffmann heuristic, both of which are used to obtain upper bounds. Section 2.3 summarises the preprocessing step by combining the components described in Section 2. In Section 3, we describe the branch-and-bound procedure in further detail: the enumeration procedure is described in Section 3.1, the bounding strategies during the search tree are introduced in Section 3.2, the dominance rules are described in Section 3.3, and the steps performed by the branch-and-bound algorithm are schematised in Section 3.4. Section 4 presents the results obtained in the computational experience with the established benchmark set of problem instances. Finally, Section 5 gives a brief summary and some conclusions. Table 1 shows the notation that will be used throughout the paper.

2. Problem preprocessing

2.1. Bounding, reduction techniques and logical tests

2.1.1. Lower bounds for SALBP-1

Our procedure uses a well-known set of seven lower bounds, which will be referred to using Scholl's notation [14]. These lower bounds will be used to prove whether a known solution is optimal and to preprocess the problem. Their use in the bounding strategy inside the branch-and-bound procedure will be further described in Section 3.2.1.

Four of these bounds are an assimilation of the SALBP-1 problem to a BPP-1. The first and most intuitive bound (LB_1) neglects the precedence and integrality constraints; it can be defined as the rounded-up sum of all of the operation times divided by the cycle time. The second bound (LB_2) divides tasks into groups by their operation times: the first group (J_1) includes all tasks j that verify $t_j > c/2$. The number of tasks in this group is a bound itself because none of these tasks may share a station with another task included in J_1. The bound may be strengthened by adding half the number of tasks included in J_2, a second group composed by tasks with an operation time of exactly half the cycle time.

A third bound (LB_3) can be defined similarly to LB_2 by dividing tasks into groups considering operation times in thirds of cycle time. Based on the same concept of counting tasks with specific operation times, another bound can be defined [3]: LB_6 divides tasks into three groups considering halves and thirds of the cycle time, sorts them by duration and successively tries to assign them to virtual stations. Adding a minimum number of stations for the tasks that have not been assigned further restricts this bound.

A fourth bound (LB_4) can be defined by relaxing the problem to a one-machine scheduling problem [12]. By applying this bound to both the original and reversed problem, we obtain the *heads* and *tails* of every task. The earliest E_j and latest L_j stations to which each task may be assigned can be calculated given an upper bound m and the a_j (heads) and n_j (tails), respectively:

$$E_j = \lfloor a_j \rfloor + 1 \tag{1}$$

$$L_j = m - \lceil n_j \rceil + 1 \tag{2}$$

To further restrict the following bound, the latest stations may be calculated as the latest to which the task can be assigned to improve the current solution by considering a more restrictive upper bound ($m = m - 1$):

$$L_j = m - 1 - \lceil n_j \rceil + 1 = m - \lceil n_j \rceil \tag{3}$$

If any task j verifies $E_j > L_j$ for a number of stations m', then a feasible solution with m' stations cannot exist. The minimum m' for which all tasks verify $E_j \leqslant L_j$ is a fifth bound (LB_5). Our algorithm also uses the classical formulation of LB_5: for each task, the sum of h_j, z_j and processing time p_j is obtained. The largest of these values constitutes a fifth lower bound (LB_5).

Finally, a seventh lower bound (LB_7) is found by assimilating the SALBP-1 problem to a SALBP-2 (which consists of minimising the cycle time for a given number of stations). For an m' number of stations (the most restrictive lower bound), a minimum number of tasks bound to share a station can be found. This number is used to calculate an upper bound for the cycle time $c(m')$. The first m' to verify $c(m') \geqslant c$ is a lower bound to the original problem.

2.1.2. Reduction rules

The first reduction rule increases the task operation time, if possible. Known as the Extended Duration Augmentation Rule, this rule constitutes an extension of the Johnson dominance rule [12]. Considering no precedence constraints and known E_j and L_j for all tasks, the set of tasks that can share a station with a task j is composed by those tasks i for which $E_i \leqslant L_j$ and $E_j \leqslant L_i$. If no combination between j and the tasks in this set fills the station completely, t_j can be increased by the difference between the cycle time and the maximum possible load. The problem can be solved as a 0–1 knapsack problem, considering the task operation times as both weights and values and a maximum load equal to the cycle time, as observed in Martello and Toth [13].

Table 1
Notation used in the paper.

Symbol	Description
c	Cycle time
n	Number of tasks
i, j	Task identifiers
t_j	Operation time for task j
k, h	Station identifiers
k_f	Earliest unloaded station, used during the description of the branch-and-bound
k_b	Latest unloaded station, used during the description of the branch-and-bound
m	Current upper bound on the number of stations, obtained by any solving procedure
m_f	Number of stations constructed in the forward direction
m_b	Number of stations constructed in the backward direction
B_k	Set of tasks assigned to station k
B_{Pj}	Set of total predecessors for task j
B_{Fj}	Set of total followers for task j
a_j	Head for task j, obtained in the calculations for LB_4
n_j	Tail for task j, obtained in the calculations for LB_4
E_j	Earliest possible station for task j, given h_j
L_j	Latest possible station for task j, given a_j and an upper bound
T_k	Average task time for station k, used to select a branching direction
p_j	Processing time for task j, defined as $p_j = t_j/c$
D_k	Load for station k, defined as the sum of processing times of the tasks assigned to station k, $\sum_{\forall i \in B_k} t_i$