European Journal of Operational Research 240 (2015) 32–42

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Heuristics and lower bounds for the simple assembly line balancing problem type 1: Overview, computational tests and improvements

Tom Pape *

Clinical Operational Research Unit, University College London, 4 Taviton Street, WC1H 0BT London, UK

ARTICLE INFO

Article history:
Received 24 November 2012
Accepted 19 June 2014
Available online 1 July 2014

Keywords:
Heuristic
Lower bound
Assembly line balancing
Reduction technique
Partitioning problem

ABSTRACT

Assigning tasks to work stations is an essential problem which needs to be addressed in an assembly line design. The most basic model is called simple assembly line balancing problem type 1 (SALBP-1). We provide a survey on 12 heuristics and 9 lower bounds for this model and test them on a traditional and a lately-published benchmark dataset. The present paper focuses on algorithms published before 2011.

We improve an already existing dynamic programming and a tabu search approach significantly. These two are also identified as the most effective heuristics; each with advantages for certain problem characteristics. Additionally we show that lower bounds for SALBP-1 can be distinctly sharpened when merging them and applying problem reduction techniques.

1. Introduction

Assembly lines are a common way to organise mass production of standardised products. They consist of ordered stations along a conveyor belt to which a set of tasks is assigned to. The cycle time determines how much time the stations' workers and/or machines have to fulfil their tasks before passing on the workpiece to the next following station.

The simple assembly line balancing problem type 1 (SALBP-1) is a fundamental and well-studied problem of assembly line design (Baybars, 1986; Scholl, 1999). The tasks $j = 1,\ldots,n$ are defined by task times t_j and their positions within the precedence graph. The goal is to minimise m as number of loaded stations given a fixed cycle time c. A list of all used symbols can be found in Table 1. Fig. 1 illustrates SALBP-1 exemplarily. The nodes (tasks) of the precedence graph are indexed from 1 to 8 and above them stand their task times t_j. For SALBP-1 a solution is feasible if (i) the tasks of each station do not have a task time sum larger than c and (ii) no direct or indirect predecessor of any task j is assigned to a later station than j is assigned to. The shaded regions identify a possible feasible solution with 4 stations. If one turns around the arrows' directions in the precedence graph, one receives the reverse problem. A solution of the reverse problem (backward direction) is always a feasible solution of the original SALBP-1 (forward direction) after turning around the station order.

Many general assembly line balancing problems (GALBPs) base on this simple logic and extend it, for example, with ergonomic considerations, space restraints and mixed-model production. Therefore algorithms should be analysed properly on their effectiveness on SALBP-1 before adapting them to more sophisticated GALBPs. Comparing the effectiveness of procedures only with the results reported in their original papers may be distorting due to different computational environments, incomparable CPU times, or different datasets. This explains the need for thoroughly conducted comparing studies. By now those exist only for some exact procedures (Baybars, 1986; Scholl & Klein, 1999), simple algorithms (Ponnambalam, Aravindan, & Mogileeswar Naidu, 1999) and priority rules (Otto, Otto, & Scholl, 2014; Scholl & Voß, 1996).

SALBP-1 is NP-hard (Karp, 1972), so that heuristics are essential to obtain upper bounds for problems. Furthermore in order to assess the quality of found solutions, lower bounds methods are important in integer optimisation. Closing the research gap by a comparing study of upper and lower bounds for SALBP-1 is the first and main goal of this paper.

The second goal is the improvement of some already-known procedures, namely tabu search, dynamic programming, lower bound 7 and 8, as well as SALBP-1 reduction techniques. It will also be discussed how to use problem reduction techniques for sharpening lower bounds.

As benchmark dataset this study uses the collection of 269 instances from Scholl (1993) as well as the new systematically-generated 100-tasks and 1000-tasks problems from Otto, Otto, and Scholl (2013) denoted as SCHOLL, OTTO-100 and OTTO-1000, respectively, in the following. By now, there has not been thor-

* Tel.: +44 7440153248.
E-mail address: t.pape@ucl.ac.uk

http://dx.doi.org/10.1016/j.ejor.2014.06.023
0377-2217/

Table 1
Symbols for SALBP-1.

c	Cycle times
j	Index of the tasks
$J(a, b]$	Set of all tasks with $a < p_j \leqslant b$
k	Index of the stations
m	Number of stations
n	Number of tasks
p_j	t_j/c_j
$\left(P_j^*\right)P_j$	(Direct) predecessors of j
$\left(S_j^*\right)S_j$	(Direct) successors of j
S_k	Load of station k
t_j	Task time of j

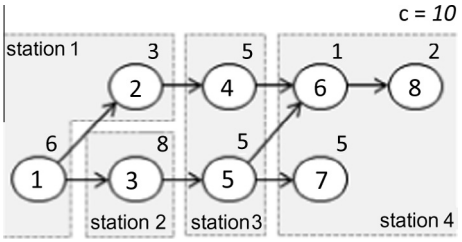

Fig. 1. SALBP-1 with a grey-shaded solution.

oughly investigated in how far the success of SALBP-1 algorithms depends on the problem properties. Finding an answer to this question is the paper's third goal.

The article is organised as follows: Section 2 outlines the "General idea" of each examined heuristic briefly, states some "Experience" the author made during implementation and describes methodical improvements for some approaches. The "General ideas" require some knowledge about standard solution procedures in operational research and the "Experience" can usually not be fully understood without having read the original papers before. Section 3 explains the improvements proposed for lower bounds. Section 4 reports the computational result. Section 5 summarises and discusses the main findings.

2. Heuristics

2.1. Falkenauer and Delchambre (1992): GA-FD

General idea: Falkenauer and Delchambre designed a genetic algorithm (GA) in which the genes are the station loads of the solution (chromosome). Thereby the genes on the chromosome are not necessarily ordered in the sequence of the stations in the final solution. Instead the precedence relations between the genes are kept in an additional genes' precedence graph. This encoding technique is called group encoding. Falkenauer and Delchambre apply some of the usual genetic operators on the genes to obtain improved children. Thereby they use a fitness function which rewards well-filled stations more than it punishes less filled ones in a solution. After crossover the children become usually infeasible since tasks are assigned to more than one station and the precedence relations are violated. A complex healing process must follow therefore.

Experience: Falkenauer and Delchambre's proposed strategy to make children feasible (eliminating cycles in the genes' precedence graph) needed often more than 500,000 iterations (>1 minute CPU time) on OTTO-100 just to obtain one new solution. In our experiments, the time limit was often reached before repairing the children of the first crossover.

2.2. Sabuncuoglu, Erel, and Tanyer (2000): GA-S

General idea: Sabuncuoglu et al. introduce a genetic algorithm in which the tasks as genes are always ordered on the chromosome in a sequence that obeys the precedence graph (order encoding). They apply a crossover technique which completely avoids infeasibility. In contrast to GA-FD, chromosomes with equally loaded stations receive the highest fitness score. Additionally they apply a freezing technique which finalises the task assignments of the current first and last unfrozen station if they provide satisfying loads. It shall be noted that a better performing hybrid genetic algorithm incorporating priority rules and local search is published in Gonçalves and De Almeida (2002).

Experience: Due to the smart crossover technique, GA-S is fast in creating new solutions (about 5000 per second on OTTO-100). The freezing often leads to a search stop before reaching an optimal solution or the given time limit. In those cases we revoke the algorithm with a twice that strict freezing policy. After preliminary tests we chose a population size of 100, a mutation probability of 1%, a final replacing probability of 1%, and an initial freezing parameter DPC of 10% which halves itself every time all stations are frozen.

2.3. Nearchou (2005): DEA

General idea: Nearchou designed a differential evolutionary algorithm (DEA) to tackle SALBP-1. We assume that learning about the tasks' variability concerning their positions in good solutions is its main idea. The solutions (chromosomes) are sub-range encoded what can be linearly transformed into order encoding. Sub-ranges express an order position of a task by small float-point intervals between 0 and 1. For instances the solution $(0.4, 0.32, 0.7)$ basing on the sub ranges $[0, \frac{1}{3}) \rightarrow 1$, $[\frac{1}{3}, \frac{2}{3}) \rightarrow 2$ and $[\frac{2}{3}, 1) \rightarrow 3$ would mean an order encoding $(2, 1, 3)$. Crossover works like in genetic algorithms, just mutation – which is performed in every iteration – is done differently. Given three sub-range encoded solutions x_a, x_b, x_c of the population as vectors, one receives the mutant with $x_m = x_c + w(x_a - x_b)$ where w denotes a small weight. Solutions after mutation and crossover need to be healed of infeasibility.

Experience: Repairing infeasible solutions makes up 85% of the CPU time and leads to a low number of created solutions per second (about 60 per second on OTTO-100). For the parameters w, crossover probability and population size we opted for 0.3, 1 and 100, respectively, after some tests. Nearchou demands a replacement of some solutions when the population becomes too homogeneous but does not define how he measures homogeneity. Therefore, we replace one solution from the bottom half of the population space according to their fitness with a new random one after every five iterations.

2.4. Bautista and Pereira (2002): ACO

Bautista and Pereira test several versions of ant colony optimisation (ACO) for SALBP-1. Here only Bautista and Pereira's best version (task-position policy, summed trail reading) is described and implemented. Very similar designs can be found in Boysen and Fliedner (2008) and Zhang, Cheng, Tang, and Zhong (2007). An ant colony algorithm which strongly relies on a local search is published in Bautista and Pereira (2007).

General idea: Solutions are order encoded, and between each task j and each order position o exists a pheromone trail τ_{jo}. Solutions are constructed either in forward or in backward direction by adding one task after another. To select the next task for order position $\bar{o}$ from all those without unassigned predecessors, a roulette wheel selection based on the task preferences is conducted. The task preference is the weighted product of the pheromone trails from order position 1 to $\bar{o}$, i.e. $\sum_{o=1}^{\bar{o}} \tau_{jo}$, and a normalised task