European Journal of Operational Research 236 (2014) 403–409

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset

David R. Morrison [a,*], Edward C. Sewell [b], Sheldon H. Jacobson [a]

[a] University of Illinois, Urbana-Champaign, Dept. of Computer Science, 201 N. Goodwin Ave., Urbana, IL 61801, United States
[b] Southern Illinois University Edwardsville, Dept. of Mathematics and Statistics, Edwardsville, IL 62026, United States

ARTICLE INFO

Article history:
Received 15 November 2012
Accepted 24 November 2013
Available online 4 December 2013

Keywords:
Assembly line balancing
Branch and bound
Combinatorial optimization

ABSTRACT

The simple assembly line balancing problem (SALBP) is a well-studied NP-complete problem for which a new problem database of generated instances was published in 2013. This paper describes the application of a branch, bound, and remember (BB&R) algorithm using the cyclic best-first search strategy to this new database to produce provably exact solutions for 86% of the unsolved problems in this database. A new backtracking rule to save memory is employed to allow the BB&R algorithm to solve many of the largest problems in the database.

1. Introduction

The assembly line balancing problem is a well-studied problem with many applications, including the automotive industry, consumer electronics, and household items (Baybars, 1986; Sarker & Pan, 2001). This problem has many variants with different objectives and side constraints; see Battaïa and Dolgui (2013) for a recent survey of problem formulations and solution techniques. One of the most basic assembly line balancing problems is the Simple Assembly Line Balancing Problem (SALBP). In this problem, a set of tasks $T = \{1, 2, \dots, n\}$ is given that must be accomplished by a set of workers or stations. In many applications, stations are designed to complete specific tasks; however, the SALBP relaxes this assumption so that all stations are considered identical. Each task requires a certain amount of time t_j (called the **processing time**) to complete, and each station has a specified fixed amount of time c (called the **cycle time**) that it can spend completing tasks.

Additionally, a directed acyclic graph G, called the **precedence graph**, is given with vertex set T and arc set A. An arc $(i, j) \in A$ indicates that task i must be completed before task j. A task i is a **predecessor** (alternately, **successor**) of j if there is a path from i to j (alternately, from j to i) in G; if this path has length 1, i is a **direct** predecessor or successor. The set of direct predecessors (successors) of j is denoted $P_j(F_j)$, and the set of predecessors (successors) of j is $P_j^*\left(F_j^*\right)$.

The objective of SALBP is to find the minimum number of stations needed to complete all tasks, subject to the cycle time at each station, that satisfies all relations given in the precedence graph. Given a set of tasks S_m assigned to the m^{th} station, define the **idle time** I_m as the amount of time the station is not working; that is, $I_m = c - \sum_{j \in S_m} t_j$. For a complete assignment of tasks to stations, the total idle time I is the sum of the idle times at each station.

Despite the fact that SALBP is NP-complete (the bin-packing problem is a special case where G has no edges), a number of well-known exact algorithms exist for solving SALBP. An early algorithm developed called EUREKA (Hoffmann, 1992) used an effective heuristic together with a branch-and-bound algorithm that explored in both the forward and reverse directions (i.e., by assigning tasks to either the first stations first or the last stations first). Extensions to the Hoffman heuristic were proposed in Fleszar and Hindi (2003) that were able to perform quite well on a subset of standard benchmark problems.

Johnson (1988) describes an algorithm called FABLE which incorporates a number of bounding rules and dominance rules for the SALBP problem; additionally, Nourie and Venta (1991) give an algorithm called OptPack which uses a dominance rule called the **tree dominance rule**. Another branch-and-bound algorithm, called SALOME (Scholl & Klein, 1997, 1999), also incorporated a bi-directional search strategy together with several highly effective lower bounds, dominance rules, and a new branching strategy. Another lower bound, based on an LP relaxation of SALBP, is described in Peeters and Degraeve (2006). A dynamic-programming heuristic for SALBP is given in Bautista and Pereira (2009) that incorporates bounding mechanisms into the DP table. Finally, Scholl and Becker (2006) provide a comprehensive survey of SALBP, discussing various bounds and solution methods (both exact and heuristic).

More recently, Sewell and Jacobson (2012) give a highly-effective algorithm for SALBP that is able to solve all 269 test instances in a list of benchmark instances, including one instance that had previously been open for over a decade. This algorithm incorporates a three-phase solution procedure together with the cyclic

* Corresponding author. Tel.: +1 217 244 7275; fax: +1 217 244 6869.
E-mail address: drmorr0@gmail.com (D.R. Morrison).

best-first search (CBFS) strategy, as well as a number of good lower bounds and a memory-based dominance rule. A new set of benchmark instances, as well as an instance generator called SALBPGen, was subsequently released by Otto, Otto, and Scholl (2013); this dataset contains 6825 problem instances, ranging in size from small (20 tasks) to very large (1000 tasks), and incorporates a number of features commonly seen in real-world data.

This paper extends the algorithm of Sewell and Jacobson (2012) with a new backtracking procedure for very large problem instances, and presents computational results on the new dataset of Otto et al. (2013). In this new dataset, 1359 instances are listed as unsolved; the most significant contribution of this paper is to demonstrate that 1172 instances (i.e., all but 187 instances) can each be solved in under 1 hour of computation time, and a further 184 have the best-known solution improved upon. Moreover, all of the previously-solved instances in this database are also solved by this algorithm. Additionally, a proof of compatibility between the dominance rules used in this algorithm is provided. Finally, a brief study of the remaining unsolved instances is also performed to determine what characteristics make them challenging for the Sewell and Jacobson (2012) algorithm.

This paper is organized as follows: Section 2 provides a description of the new SALBP problem database. Section 3 describes the branch, bound, and remember algorithm used to solve these problems, and Section 4 presents the suite of computational results performed against the instances in the database. Finally, Section 5 gives some concluding remarks.

2. Problem testbed description

The SALBPGen algorithm of Otto et al. (2013) was designed to emulate properties seen in real-world assembly lines, particularly from the automotive industry. In particular, two graph properties were identified that commonly appear in the precedence graph G for these problems; these are **bottleneck tasks** and **chains**. A third property is that of **modular** design, which is an optional additional generation parameter that groups nodes into related clusters or modules, and builds a super-precedence graph on the modules. However, the modular design option is not used in the benchmark dataset.

A bottleneck task j has high in- and out-degree in G; furthermore, it is the only direct successor for at least two tasks in P_j, and it is the only direct predecessor for at least two tasks in F_j. A chain of tasks, on the other hand, is a set of tasks $C \subseteq T$ with $|C| \geqslant 2$ such that C forms a path in G and $|P_j| = |F_j| = 1$ for each $j \in C$.

Another important property of SALBP instances is the **order strength**; this value, denoted by OS, is computed as $|A(G^+)|/\binom{n}{2}$, where $A(G^+)$ is the arc set of G^+, the **transitive closure** of G. That is, G^+ is the graph with vertex set T where arc (i,j) indicates that task i is a (not necessarily direct) predecessor of task j. As stated in Scholl and Klein (1999), "Small values of OS indicate that the precedence constraints are not very restrictive such that many sequences of tasks are feasible." There are some indications that middle values of OS are harder than low or high order strength values. The generator SALBPGen allows an input parameter to be given specifying the desired order strength of the graph.

A third important parameter that can be controlled by SALBPGen is the distribution of task times; for each instance, task times are randomly generated according to some pre-specified probability distribution. The problem database contains instances with task times that have been generated according to three different distributions, described below:

- Short task time distribution-task times are drawn from a normal distribution with the mean centered around small times.
- Bimodal task time distribution-task times are drawn from a combination of two normal distributions with means centered around small and large times.
- Centralized task time distribution-task times are drawn from a normal distribution with a mean task time of $c/2$.

The first two task time distributions emulate properties seen in real-world instances of the assembly line balancing problem; the latter is designed to produce challenging instances.

The problem database used for testing in this paper was generated and described in Otto et al. (2013). The database contains instances with $n = 20$, 50, 100, and 1000 tasks (called small, medium, large, and very large, respectively). There are 525 instances of each problem size, which have been generated with varying order strengths and distribution of task times. A third of the problems (called BN instances) have been generated with bottleneck nodes having minimum degree eight (or minimum degree four in the small instances). A third (called CH instances) have been generated with 40% of the nodes in chains, and a third of the instances (called MIX instances) have no such requirements on the structure of the precedence graph.

For each problem instance in the medium dataset, there are 9 additional permuted instances, which share a common precedence graph and set of task times, but have randomly assigned the task times to tasks. Thus, there are a total of 6825 instances in the dataset. Of these instances, Otto et al. (2013) report that 4 small instances, 846 medium instances (including permutations), 170 large instances, and 339 very large instances have not yet been solved, for a total of 1359 unsolved instances. No other papers were found in the literature that have improved upon these results thus far.

3. The BB&R algorithm for SALBP

Algorithm 1. BBR(t, G, c)

```
1 ComputeDirection($t, G$)
2 $UB$ = ModifiedffmannHeuristic($t, G, c$)
3 $LB_{\text{root}} = \max(LB1, LB2, LB3, BPLB)$ ⟨⟨Global lower bound is
  best lower bound at the root⟩⟩
4 $UB$ = RunSearch($UB, LB_{\text{root}}$, CBFS)
5 if $\tau < \tau_{lim}$ and previous search was not (provably) optimal:
6    $UB$ = RunSearch($UB, LB_{\text{root}}$, BrFS)
```

Algorithm 2. RunSearch$(UB, LB_{\text{root}}, \text{mode})$.

```
1 $c = 0$
2 root = $(\emptyset, T)$ ⟨⟨The root node has no assigned tasks to any
  station⟩⟩
3 $X$ = root
4 While search tree is non-empty, $c < n_{lim}$, $\tau < \tau_{lim}$, and
  $LB_{\text{root}} < UB$:
5    if $\max(LB1_X, LB2_X, LB3_X, BPLB_X) \leqslant UB$ or $X$ is dominated:
  rune $X$
6    else if $X$ dominates another subproblem $Y$: delete $Y$
7    else if $X$ is terminal $UB = \min(m, UB)$
8    else:
9       for each valid $S_{m+1}$, given $A_X$:
10          $Y = \left(A_X \cup S_{m+1}, U_X \setminus S_{m+1}, S_1^X, S_2^X, \ldots, S_m^X, S_{m+1}\right)$
11          Insert $Y$ into search tree and increment $c$
12          ⟨⟨If too many subproblems are generated at a node,
  the search continues in a heuristic manner⟩⟩
13          if more than $s_{lim}$ subproblems have been generated
  from $X$:
14             stop generating subproblems
15    Select a new $X$ according to the mode (CBFS or BrFS)
16 return UB
```