

Can a Branch and Bound algorithm solve all instances of SALBP-1 efficiently?

Alexandre Dolgui* Evgeny Gafarov**

* *IMT Atlantique, LS2N, CNRS, 4, rue Alfred Kastler, 44307 Nantes cedex 3, France (email: alexandre.dolgui@mines-nantes.fr)*

** *V.A. Trapeznikov Institute of Control Sciences of the Russian Academy of Sciences, Profsoyuznaya st. 65, 117997 Moscow, Russia. Tel.: +7 925 809 09 07. (email: axel73@mail.ru)*

Abstract: Following the publication by Sewell and Jacobson of a Branch and Bound algorithm which has solved all existing benchmarks for the simple assembly line balancing problem of type 1 (SALBP-1) in less than 1 second per problem, this paper investigates the possible existence of harder instances and how they can be designed. SALBP-1 is a generalization of the bin packing problem. We demonstrate how a hard bin packing case can be generated. We prove that, in the worst case, Branch and Bound algorithms with Lower Bounds computed in polynomial time are unable to solve even moderate sized examples of the problem in appropriate time limit. Branch and Bound is an excellent approach based on decomposition and enumeration. Intelligent techniques were elaborated to avoid a complete enumeration. Nevertheless, as demonstrated here, it is always possible to find cases where a Branch and Bound will necessitate a complete enumeration.

© 2019, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.

Keywords: Assembly line balancing, Bin packing, Optimization, Worst case analysis, Generation of hard instances

1. INTRODUCTION

The motivation of this paper deals with the fact that after the publication by Sewell and Jacobson of a Branch and Bound algorithm which has solved all existing benchmarks for the simple assembly line balancing problem of type 1 (SALBP-1) in less than 1 second per problem, it is interesting to know if there exist harder instances and how they can be designed.

SALBP-1 is a generalization of the Bin Packing Problem (BPP). We demonstrate how we can generate a hard bin packing case. We prove that, for the worst case, Branch and Bound algorithms with Lower Bounds computed in polynomial time are unable to solve even moderate sized examples of the problem in an appropriate time.

The work focuses on SALBP-1 because of the aforementioned motivation, nevertheless, it is also valid for the BPP and all its other generalizations.

The SALBP-1 seeks to find an optimal assignment of tasks (operations) to workstations for a given cycle time c in such a way that the number m of workstations is minimized, see Baybars (1986); Scholl (1999); Dolgui and Proth (2010).

The Branch and Bound algorithm proposed by Sewell and Jacobson (2012) is able to solve all well-known and widely employed benchmark instances for SALBP-1 in less

than 1 second each. Its recent modifications by Morrison et al. (2014) were able to solve all unsolved small and medium sized instances from the new benchmark database presented by Otto et al. (2013), and over 90% of the large instances and improve the best-known solution for other large instances from this new set of benchmarks.

Nevertheless, SALBP-1 is NP-hard (in the strong sense), see Baybars (1986), thus for each possible optimization algorithm there are cases that require prohibitive running time. *How to design such cases for the algorithm in Morrison et al. (2014) and other Branch and Bound algorithms developed for SALBP-1?* This question is addressed below.

Branch and Bound algorithms are extremely useful and effective, see Sprecher (1999). If they are stopped in the situation when the calculation time becomes prohibitive, they furnish good approximate or even optimal solutions but without proving optimality. The objective of this paper is not to criticize Branch and Bound approaches, but to highlight the limits of these algorithms.

The definition and a fundamental analysis of SALBP-1 is done in Baybars (1986). Some comprehensive surveys on SALBP-1, other versions of SALBP and their generalizations are presented, e.g., in Ghosh and Gagnon (1989); Erel and Sarin (1998); Rekiek et al. (2002); Battaia and Dolgui (2013); Hazir et al. (2015).

SALBP-1 is formulated as follows. A set $N = \{1, 2, \dots, n\}$ of operations is given. For each operation $j \in N$, a processing time $t_j \geq 0$ is defined. A cycle time $c \geq$

* The work was partially supported by the region Pays de la Loire, France, and by RSF (Russian Science Foundation): 17-19-01665.

$\max\{t_j, j \in N\}$ is also fixed. Furthermore, the finish-start precedence relations $i \rightarrow j$ are defined between the operations according to an acyclic directed graph G . The objective is to assign each operation $j, j = 1, 2, \dots, n$, to a station in such a way that:

- number m of stations used is minimized;
- for each station $k = 1, 2, \dots, m$, its total workload $\sum_{j \in N_k} t_j$ does not exceed c , where N_k is the set of operations assigned to station k ;
- precedence relations are fulfilled, i.e. if $i \rightarrow j, i \in N_{k_1}$ and $j \in N_{k_2}$, then $k_1 \leq k_2$.

2. DESIGN OF A HARD CASE

In this section, a specific case of BPP, and consequently of SALBP-1, is presented and we prove that all Branch and Bound ($B\&B$) algorithms with any polynomial time computed lower bound cannot solve even moderate instances in a short time.

First, we present some known results for classical Knapsack and Partition Problems.

Partition problem. A set $N = \{b_1, b_2, \dots, b_n\}$ of values $b_1 \geq b_2 \geq \dots \geq b_n > 0$ with $b_i \in \mathbb{Z}_+, i = 1, 2, \dots, n$, and a value $A \in \mathbb{Z}_+$ with $A < \sum_{j \in N} b_j$ are given. Is there a subset $N' \subset N$ such as $\sum_{j \in N'} b_j = A$?

The complexity of Branch and Bound for combinatorial optimization problems was considered in Aardal and Lenstra (2004); Jeroslow (1974); Krishnamoorthy (2008). The worst case running time of $B\&B$ algorithms for the well-known Knapsack Problem was analyzed, e.g., in Finkelstein (1976); Kolpakov and Posypkin (2010). The authors proposed special cases for which, if a $B\&B$ algorithm is used, then to prove the optimality of a solution, almost all feasible solutions should be considered, i.e. in each case, the computational time of the $B\&B$ algorithm would be prohibitive.

Remember that all $B\&B$ algorithms are based on the following idea. There is an enumeration procedure which is called branching. It constructs a search tree which represents ALL feasible solutions, e.g., all feasible solutions with maximal loaded stations. Different methods are used to reduce the search tree. Computing of lower bounds and comparing them with the best solution found is one of these methods.

In Posypkin and Sigal (2006), a special case of the Knapsack Problem was presented. For this case, all $B\&B$ algorithms¹ had an exponential number of nodes in a search tree, unless $P = NP$, i.e. the running time of all $B\&B$ algorithms is exponential and approximately equal to $\frac{3}{2} \cdot \frac{2^{n+3/2}}{\sqrt{\pi(n+1)}}$.

For this case, there is a sub-problem which is an instance of the Partition Problem, i.e. to compute a "useful" upper bound, the corresponding Partition Problem has to be solved, which is NP-hard. Consequently, if a polynomial upper bound is used, the pruning of nodes by the rule "a local upper bound \leq the best known value", where the

"best known value" is the maximal value of $f(x)$ already found, will be ineffective.

A similar idea will be employed to construct a special case of BPP for SALBP-1 for which any $B\&B$ algorithm has an exponential running time no matter what polynomial time lower bound is used.

The following reduction from the Partition Problem to the special case of SALBP-1 is used.

Modified instance of the Partition problem. There are a set $\bar{N} = \{\bar{b}_1, \bar{b}_2, \dots, \bar{b}_{2n}\}$ of values $\bar{b}_1 \geq \bar{b}_2 \geq \dots \geq \bar{b}_{2n} > 0$ with $\bar{b}_i \in \mathbb{Z}_+, i = 1, 2, \dots, 2n$, and a value $\bar{A} \in \mathbb{Z}_+$ with $\bar{A} < \sum_{j \in \bar{N}} \bar{b}_j$. The values $\bar{b}_i, i = 1, 2, \dots, 2n$ are calculated as follows:

$$\begin{cases} \bar{b}_{2n} = A, \\ \bar{b}_{2i} = 2 \cdot \sum_{j=i+1}^n \bar{b}_{2j}, i = 1, \dots, n-1, \\ \bar{b}_{2i-1} = \bar{b}_{2i} + b_i, i = 1, \dots, n, \end{cases} \quad (1)$$

where the values b_1, b_2, \dots, b_n are given in the initial instance. Let $\bar{A} = \sum_{i=1}^n \bar{b}_{2i} + A$.

For example: $N = \{15, 10, 9, 3\}; A = 24$; then ${}^2 \bar{N} = \{146 + 15 = 161; 2(52 + 17 + 4) = 146; 42 + 10 = 52; 2(17 + 4) = 42; 8 + 9 = 17; 2 \cdot 4 = 8; 1 + 3 = 4; 1\}$ and $\bar{A} = 1 + 8 + 42 + 146 + 24 = 221$.

Without loss of generality, let us assume $A = \frac{1}{2} \sum_{i=1}^n b_i$ and, as consequence, $\bar{A} = \frac{1}{2} \sum_{i=1}^{2n} \bar{b}_i$.

The question is: "Is there a subset $\bar{N}' \subset \bar{N}$ such that $\sum_{j \in \bar{N}'} \bar{b}_j = \bar{A}$?" It is obvious that this modification can be done in polynomial time. If, for the initial instance of the Partition Problem the answer is "YES" (the modified instance has the same answer), then \bar{N}' contains one and only one value from each pair $\{\bar{b}_{2i-1}, \bar{b}_{2i}\}, i = 1, 2, \dots, n$.

We can easily prove that \bar{N}' contains one and only one value from each pair $\{\bar{b}_{2i-1}, \bar{b}_{2i}\}, i = 1, 2, \dots, n$, by induction. It holds for pair $\{b_1, b_2\}$ since their sum is bigger than the sum of all other numbers from \bar{N} . Let it hold for $i \in \{2, \dots, n-1\}$. Since $b_{2(i+1)-1} + b_{2(i+1)} > 2 \sum_{j=i+2}^n b_{2j} + A$ then \bar{N}' contains only one of $\{b_{2(i+1)-1}, b_{2(i+1)}\}$.

If the value b_i is included in the set N' , then \bar{b}_{2i-1} is included in \bar{N}' , otherwise the value $\bar{b}_{2i} \in \bar{N}'$.

This reduction using pairs $\{\bar{b}_{2i-1}, \bar{b}_{2i}\}, i = 1, 2, \dots, n$ was done in order to make the number of branches in a $B\&B$ algorithm exponential.

Let us consider a special case of SALBP-1 with $2n$ operations. Let $w' = \min\{w | 10^w \geq 2\bar{A}\}$. Processing times of operations are defined as follows:

$$t_i = 10^{w'} + \bar{b}_i, i = 1, 2, \dots, 2n.$$

¹ upper bounds computed in polynomial time are only used

² See the list from the end.

Furthermore, $c = \frac{1}{2} \sum_{i=1}^{2n} t_i$. There are no precedence relations between operations. As previously stated, SALBP-1 without precedence relations corresponds to the BPP which minimizes the number of bins considering the bin capacity.

This reduction using a big value w' was done in order to make a lower bound based on pseudo-polynomial time dynamic programming ineffective.

Therefore, if and only if for the modified instance of the Partition Problem the answer is "YES", then the minimal number of stations for this case of SALBP-1 is $m^* = 2$ and total workload of both machines equals c , otherwise $m^* = 3$. As a consequence, if $NP \neq P$, there is no polynomial time computed Lower Bound with a relative error less than $\frac{3}{2}$. This means that for any set of polynomial time computed lower bounds $\{LB_1, LB_2, \dots, LB_X\}$, there is a modified instance of the Partition Problem with an answer "NO", for which $LB_x = 2$, $i = 1, 2, \dots, X$, although $m^* = 3$. For this instance, any feasible solution is optimal. However, to prove its optimality almost all feasible solutions must be considered.

3. COMPLEXITY OF THE HARD CASE

Let us estimate the possible number of feasible solutions. On the first station there could be at least $n-1$ operations. Thus, there are at least $\binom{2n}{n-1}$ possible loads for the first station, i.e. the number of feasible solutions which have to be considered is greater than:

$$\binom{2n}{n-1} = \frac{n+1}{n} \binom{2n}{n} \approx \frac{n+1}{n} \cdot \frac{2^{2n}}{\sqrt{n\pi}}.$$

To solve this instance of BPP for SALBP-1 with 60 assembly operations ($2n = 60$) a computer must perform more than $\frac{2^{60}}{11}$ computer operations. Let us assume that the fastest known computer performs 2^{32} operations per second³, or less than 2^{48} operations per day. Then the running time of any algorithm will be more than $\frac{2^{12}}{11} > 372$ days! This means that there are instances of SALBP-1 for which no *B&B* algorithm with polynomial lower bounds will have an acceptable running time. Obviously, a decision maker will not wait longer than some hours for a solution (even less, usually a psychological barrier of ten minutes is considered as the maximum of an acceptable time). The best computer performances in operations per second is increasing permanently. Nevertheless, this increase cannot compensate the explosion of number of solutions.

Some *B&B* algorithms use other reducing techniques, e.g., in the Eureka algorithm by Hoffmann (1992), the author considers numbers $m = 1, 2, \dots$, one-by-one in order to find the minimal m for which there is a feasible solution (line balance) which uses only m stations. For the instance considered, for $m = 2$, we have to check $\approx \binom{2n}{n-1}$ solutions, i.e. we have the same running time as for the above described standard technique.

Now let us discuss the most recent *B&B* algorithm from Sewell and Jacobson (2012); Morrison et al. (2014). The

³ The fastest known on November 2018 computer "Summit" performs 2^{32} operations per second

following methods are used in this algorithm to reduce the search tree.

1. Lower bounds. $LB_1 = \lceil \frac{\sum t_i}{c} \rceil$, $LB_2 = |\{j \in N | t_j > c/2\}| + \lceil \frac{|\{j \in N | t_j = c/2\}|}{2} \rceil$. LB_3 is computed by assigning a weight w_j to each operation j :

$$w_j = \begin{cases} 1 & \text{if } t_j > 2c/3, \\ \frac{2}{3} & \text{if } t_j = 2c/3, \\ \frac{1}{2} & \text{if } c/3 < t_j < 2c/3, \\ \frac{1}{3} & \text{if } t_j = c/3, \end{cases} \quad (2)$$

then $LB_3 = \lceil \sum w_j \rceil$. The reader is referred to Sewell and Jacobson (2012) for a complete description. For the special case of SALBP-1 introduced in this paper, $LB_1 = 2$ and $LB_2 = LB_3 = 0$, i.e. all of these lower bounds are futile.

2. To solve the sub-problems, a Bin-Packing *B&B* algorithm is used. Nevertheless, for the special case proposed here, such an algorithm has an exponential running time as well, i.e. the search tree will not be reduced substantially.
3. The authors use a hybrid of depth-first and best-first search known as Cyclic Best First Search. If memory is exhausted, the algorithm will switch to breadth-first search. This breadth-first strategy could be inappropriate for the case suggested here, in view of the amount of memory required.
4. The authors also employ dominance rules in addition to bounding. We assume the dominance rules do not allow for sufficient pruning. In Sewell and Jacobson (2012) three rules are used. Two of them are based on precedence constraints and therefore useless for the special case. The third rule "if a partial solution is not maximally loaded, then that partial solution can be pruned" do not allow for sufficient pruning since all $\binom{2n}{n-1}$ solutions considered are maximally loaded.

Therefore, the following can be concluded. Although the algorithm of Sewell and Jacobson (2012) has solved all SALBP-1 benchmarks, known at that time, in less than 1 second each, the proposed special case is very hard and cannot be solved in a reasonable time by the algorithm assuming the dominance rules do not allow for sufficient pruning. For this case, there is a threshold for the maximal number of operations (a relatively small number of operations) after which the *B&B* algorithm is ineffective.

We can conclude that establishing new *B&B* solution procedures to solve to optimality even medium-sized instances for the proposed hard case of Bin Packing Problem appears to be unpromising. This is a very interesting theoretical result which is based on the application of worst case analysis techniques. Based on this result, if, in an SALBP-1 (or another generalization of BPP) instance, we find this

hard core case, we can conclude that *B&B* algorithms are not effective.

Note, that other known algorithms or heuristics can also be ineffective for this case.

From a practical point of view, this result is also important. In practice, BPP, SALBP-1 and its generalizations are often solved by Branch and Bound algorithms. Most of time, *B&B* are used as heuristics when they are stopped before proving optimality. Nevertheless, real life instances with hundreds (and even thousands) of operations can contain hard *core* sub-instances as suggested in this paper. The necessity to understand such hard cases and the ability to detect them to decide when to stop a Branch and Bound algorithm is an important issue.

Here, we presented one such hard core case. The search for other possible hard core cases is an interesting research perspective.

4. CONCLUSION

The recent Branch and Bound (*B&B*) algorithm by Sewell and Jacobson (2012); Morrison et al. (2014) solves all known traditional benchmarks of simple assembly line balancing problem of type 1 (SALBP-1) in less than 1 second per problem and also solves instances from the new database in Otto et al. (2013) effectively. Thus, it was interesting to discuss the question if any instance of SALBP-1 could be solved effectively with a Branch and Bound algorithm and how to design hard instances.

Branch and Bound algorithms are often used for SALBP-1. Nevertheless, the Branch and Bound approach supposes an enumeration and for some cases this enumeration needs to be exhaustive to prove optimality.

Branch and Bound is an excellent approach based on decomposition and enumeration. Intelligent techniques were elaborated to avoid a complete enumeration. Nevertheless, as demonstrated here, it is always possible to find cases where a Branch and Bound will necessitate a complete enumeration.

There are instances in which heuristics are able to produce better solutions than a Branch and Bound algorithm within some reasonable time limits. Evidently, in practice, if the calculation time becomes prohibitive, it is possible to stop a Branch and Bound algorithm. In this case, the Branch and Bound furnishes good approximates or optimal solutions but without proving optimality.

The objective of this paper was to point out this limit of *B&B* techniques.

Finally, the search for new and complementary algorithms, including exact methods, heuristics and meta-heuristics, is still an important way to solve real-life instances of BPP, SALBP-1 and their generalizations.

REFERENCES

- Aardal K. and Lenstra A., 2004. Hard equality constrained integer knapsacks, *Mathematics of Operations Research*, 29(3), 724–738.
- Battaia O., and Dolgui A., 2013. A taxonomy of line balancing problems and their solution approaches, *International Journal of Production Economics*, 142, 259–277.
- Baybars I., 1986. A survey of exact algorithms for the simple assembly line balancing, *Management Science*, 32, 909–932.
- Dolgui A., Proth J.M., 2010. *Supply chain engineering: Useful methods and techniques*, Springer, London.
- Erel E. and Sarin S., 1998. A survey of the assembly line balancing procedures, *Production Planning & Control*, 9(5), 414–434.
- Finkelstein Yu.Yu., 1976. *Approximate Methods and Applied Problems of Discrete Optimization*, Nauka, Moscow (in Russian).
- Gafarov E.R., Dolgui A., Lazarev A.A., 2012. Some Complexity Results for the Simple Assembly Line Balancing Problem, *Proceedings of the 3rd International Conference Optimization and Applications (OPTIMA 2012)*, Costa da Caparica, Portugal, 23-30 September, 81-85.
- Ghosh S. and Gagnon R.J., 1989. A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems, *International Journal of Production Research*, 27(4), 637–670.
- Hazir O., Delorme X. and Dolgui A., 2015. A review of cost and profit oriented line design and balancing problems and solution approaches, *Annual Reviews in Control*, 40, 14–24.
- Hoffmann T.R., 1992. Eureka: A Hybrid System for Assembly Line Balancing, *Management Science*, 38, 39–47.
- Jeroslow R., 1974. Trivial integer programs unsolvable by branch-and-bound, *Mathematical Programming*, 6, 105–109.
- Krishnamoorthy B., 2008. Bounds on the size of branch-and-bound proofs for integer knapsacks, *OR Letters*, 36(1), 19–25.
- Kolpakov R. and Posypkin M., 2010. Upper and lower bounds for the complexity of the branch and bound method for the knapsack problem, *Discrete Mathematics and Applications*, 20(1), 95–112.
- Morrison D.R., Sewell E.C., Jacobson S.H., 2014. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. *European Journal of Operational Research*, 236(2), 403–409.
- Otto A., Otto C., Scholl A., 2013. Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research*, 228(1), 33–45.
- Posypkin M.A. and Sigal I.Kh., 2006. Speedup estimates for some variants of the parallel implementations of the branch-and-bound method, *Computational Mathematics and Mathematical Physics*, 46(12), 2189–2202.
- Rekiek B., Dolgui A., Delchambre A. and Bratcu A., 2002. State of art of assembly lines design optimization. *Annual Reviews in Control* 26(2): 163–174.
- Scholl A., 1999. *Balancing and Sequencing of Assembly Lines*, Physica Verlag, Heidelberg.
- Sewell E.C., Jacobson S.H., 2012. A Branch, Bound and Remember Algorithm for the Simple Assembly Line Balancing Problem, *INFORMS Journal on Computing*, 24(3), 433–442.
- Sprecher A., 1999. A competitive branch-and-bound algorithm for the simple assembly line balancing problem. *International Journal of Production Research*, 37(8), 1787–1816.