Simple Assembly Line Balancing Problem (SALBP), Type 2

Aykut Bulut

May 18, 2012

Abstract

SALBP is an NP-hard problem. There are some measures proposed to grasp the difficulty of SALBP. Order Strength, Flexibility Ratio and West Ratio all of which are based on SALBP precedence networks are among them. This study investigates the sensitivity of difficulty of SALBP type 2 to the three measures. A SALBP solver is created in Python programming language. Computational experiments are conducted.

Keywords: Simple Assembly Line Balancing, Mathematical Model, SALBP Solver, Python

1 Introduction

SALBP rises in *assembly systems*. An assembly system consists of *work stations* where some specific *tasks* are carried to produce a product. Work stations are linked together by a *trasportation system*. Transportation system moves work in process from work station to work station. Work stations together with transportation system is called *assembly line*. First work station is fed by a manufacturing item and after a specific time interval workpiece at each station is transfered to the next station. This time interval is called *cycle time*. Each task has a *process time*. Each work station process tasks assigned to it within this cycle time. At the end of each cycle a product comes out of the last station. Tasks can not be assigned to stations arbitrarily. There are some technological constraints such that some tasks should has precessed before some specific task set, i.e. tasks have some *precedence relations*. These precedence relationships can be shown on a *precedence network*.

SALBP is the analytical modeling of this system under some assumptions. Assumptions given by Baybars [1] can be restated as follows.

- All input parameters are known with certainty.
- A task can not be split between two or more work stations.
- Tasks should be processed according to their precedence relationships.
- All tasks must be processed.
- Any work station can process any task.
- Task process times are fixed, and independent of work station.
- Assembly line is assumed to be serial and transportation occurs instantly. Interaction with transportation system is ignored.
- Assembly line is for a single unique product.

There are two types of SALBP, type 1 and type 2. When cycle time is fixed and objective is to minimize number of workstations required to produce the item the problem is called type 1. When number of workstations is fixed and objective is to minimize cycle time, problem is called SALBP type 2. Salveson [4] gave the first analytical statement of the SALBP and showed that SALBP type 1 is equivalent to SALBP type 2 i.e. minimizing number of work stations for a given cycle time is equivalent to minimizing cycle time for a given number of workstations. In this work we will consider SALBP type 2. Hereafter SALBP refers to SALBP type 2.

Many procedures (exact/heuristic) are proposed for solving SALBP. The following is a mixed-binary program given by Baybars in [1] that solves SALBP. Define the following parameters and variables.

Parameters
I: Number of tasks
J: Number of workstations
P_i: Set of predecessors of task i
t_i: Task time

Variables
x_{ij}: Binary variable, 1 if task i assigned to station j.
T: Cycle time

Then mixed-binary program for solution of SALBP can be given as follows.

$$\begin{aligned}
&\min\ T && && (1)\\
&s.t. && && \\
&\sum_{j=1}^{J} x_{ij} = 1 && \forall i \in \{1,2,\dots,I\} && (2)\\
&T \geq \sum_{i=1}^{I} t_i x_{ij} && \forall j \in \{1,2,\dots,J\} && (3)\\
&x_{ik} \leq \sum_{j=1}^{k} x_{hj} && \forall i \in \{1,2,\dots,I\}, \forall k \in \{1,2,\dots,J\}, \forall h \in P_i && (4)\\
&x_{ij} = 0,1 && \forall i \in \{1,2,\dots,I\}, \forall j \in \{1,2,\dots,J\} && (5)
\end{aligned}$$

(1) is the cycle time. (2) requires each task to be assigned to one workstation. (3) makes sure that cycle time is at least as large as the process time of the work station with the highest load. (4) ensures that all of the predecessors of task i is assigned to work stations either before workstation of i or to the same work station. (5) forces x_{ij} to be binary.

Aim of this study is to investigate the following propositions.

- The structure of the precedence network in the SALBP affects the difficulty of the problem as measured in resources (including time) required for solution.
- There is some measure that can quantify this difficulty apriori.

There are some studies that proposes measures for the difficulty of the problem apriori. Three of them will be investigated in this study. The first one is *ordering strength* (OR), which is proposed by Mastor, see [3]. The second one is *flexibility ratio* (FR) proposed by Dar-El, see [2]. OS is the number of precedence relations divided by the total possible number of precedence relations. Higher OS indicates more difficult problems. FR is the number of zero entries in the precedence matrix divided by the number of entries in that matrix. Precedence matrix is the matrix representation of the precedence network. FR is noted as the converse of OS (1.0-OS).

The third measure that also measures the difficulty of the SALBP is *West Ratio* (WR) proposed by Dar-El [2]. WR is the number of tasks per station in an optimal solution. Note that for SALBP type 1 WR can only be obtained after the problem is solved. Since only SALBP type 2 is the only concern in this study, WR is an apriori measure.

It is known that SALBP is NP-hard, [1]. This classification is for the worst case. In this study we will investigate the sensitivity of the difficulty of SALBP instances to the measures calculated from these instances. This investigation will lead us to an understanding about how well the measures do. The following is the research plan.

- Create a platform that can efficiently create SALBP instances, analyze precedence relations, analyze difficulty measures and solve the problem created.
- Conduct computational experiments using the tool created.
- Interpret results.

A SALBP solver is produced. Next section will introduce the solver produced. Section 3 explains the experimental parameters and gives the computational results. Finally Section 4 gives the deductions made from computational results.

2 SALBP Solver

Salbp is a Python class that is designed to create, analyze and solve SALBP. It uses different tools to achieve all of this. These tools are PuLP v1.5.1, CBC v2.7, Graphviz v2.26.3 and Pydot v1.0.28.

PuLP is an open-source linear programming (LP) modeler (continuous, integer, binary and mixed) written in Python. It enables to build LP models within Python environment. Models built in Python environment can be saved in MPS or LP format. PuLP can call different solvers to solve the problems built. Salbp inherits PuLP's LP problem class and extends it to keep additional features of SALBP problem.

Among all the solvers available for PuLP, CBC (Coin-or Branch and Cut) is used in this study. CBC is an open-source mixed integer programming solver written in `C++`. It can be used as a callable library or using a stand-alone executable. In our case PuLP will call the pre-compiled CBC executable to solve the model built.

PuLP and CBC are enough to built and solve the SALBP. Salbp class comes with additional capabilities other than creating and solving SALBP. It can process the precedence network, create graph of it and save it as a picture of the desired format. Salbp uses Graphviz and Pydot to achieve this. Graphviz is an open-source graph visualization software. It has its own language called Dot language. Salbp uses graphviz not with Dot language bu with using Pydot, which is a Python interface for Graphviz's Dot language. Pydot is also an open-source software. Salbp uses Pydot and Graphviz to visualize the precedence relationships and save it as a file.

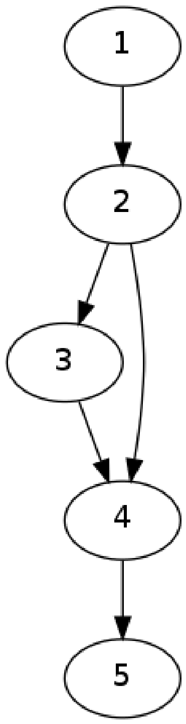
User should define cycle time, number of work stations, task times and precedence relations through Salbp's interface. Salbp builds PuLP model from this information and solves with a function call. The following is a small example of how to use Salbp.

```
from salbp import Salbp
if __name__ == '__main__':
    p = Salbp()
    p.assign_number_of_workstations(3)
    p.assign_task_time(1,3.0)
    p.assign_task_time(2,2.0)
    p.assign_task_time(3,2.0)
    p.assign_task_time(4,3.0)
    p.assign_task_time(5,3.0)
    p.assign_predecessor(2,1)
    p.assign_predecessor(3,2)
    p.assign_predecessor(4,2)
    p.assign_predecessor(4,3)
    p.assign_predecessor(5,4)
    p.save_precedence_graph('precedence_graph.png', 'png')
    p.salbp_solve()
    p.writeSALBP('salbp.lp')
```

This script first creates a SALBP instance named "p". Then assigns cycle time as 7.0. Assigns number of workstations as 3. Task time of the task 1, 2, 3, 4 and 5 are assigned as 3.0, 2.0, 2.0, 3.0 and 3.0 respectively. It assigns task 1 as a predecessor of task 2, 2 as a predecessor of 3, and so on. Saves precedence network on a file called "precedence_graph.png" in PNG format and solves the problem. Writes the built model in LP format to file "salbp.lp".

Figure 1 shows the "precedence_graph.png" produced by the given script.

Figure 1: Precedence network for the given script



There is a SALBP library on web-site `http://alb.mansci.de/`. This site contains a collection of benchmark SALBP problems that are used by researchers. Salbp has a method to read the input format of the problems obtained from this site. This provides downloaded test problems to be experimented easily. The following script reads test problem “MANSOOR.IN2” and solves it. Test problems can be obtained from `http://coral.ie.lehigh.edu/~aykut/software`.

```
from salbp import Salbp
if __name__ == '__main__':
    p = Salbp()
    p.read_input('./test_problems/MANSOOR.IN2')
    p.assign_number_of_workstations(10)
    p.salbp_solve()
```

Salbp and its documentation can be obtained from `http://coral.ie.lehigh.edu/~aykut/software`.

3 Experiments

Computational experiments are conducted using the benchmark problems from the site mentioned. OR, FR and WR are calculated for the test problems. CPU time and number of iterations that took to solve the problems are measured. Table 3 presents the results.

Problem	optimal	best found	gap	nodes	iter.	CPU time	OS	FR	WR
BUXEY	34	34	-	100	15739	11.45	0.0887	0.9113	2.9
HAHN	1775	1775	-	34	9105	16	0.0595	0.9405	5.3
LUTZ1	1526	1526	-	464	32885	17.01	0.0766	0.9233	3.2
SAWYER30	34	34	-	1866	86325	23.98	0.0735	0.9264	3.0
GUNTHER	50	50	-	2384	100972	28.58	0.0756	0.9244	3.5
KILBRID	56	56	-	300	27344	33.42	0.0626	0.9374	4.5
TONGE70	352	352	0.00	361618	8553176	3602	0.0356	0.9644	7.0
ARC111	15040	15047	0.00	276522	6662970	3600	0.0288	0.9712	11.1
WARNECKE	155	156	0.01	297587	8159778	3600	0.0423	0.9577	5.8
WEE-MAG	150	151	0.01	348031	8532206	3601	0.0314	0.9686	7.5
MUKHERJE	424	424	0.01	84956	6427557	3600	0.0414	0.9586	9.4

Gap stands for the optimality gap in percentage and iter. stands for number of iterations. Note that not all the problems are solved to optimality but some are stopped by an hour time limit. Tonge, Arch111, Warnecke, Wee-Mag, Mukherje could not solved to optimality in an hour. Figure 2 shows CPU time versus FR. It can be observed that CPU time increases as FR increases. There is a big jump in CPU time. This is due to the hard problems that can not solved less than in an hour. Figure 3 shows CPU time versus WR. Note that there is an easy problem with 5.3 WR and a hard problem with 5.3. Even though WR difference is relatively small there is a huge jump in CPU time.

Figure 2: CPU time vs FR

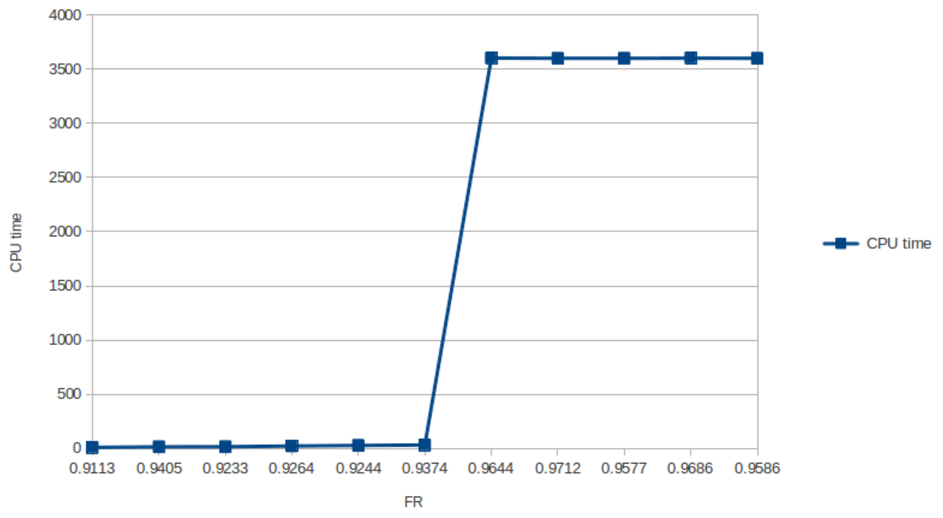
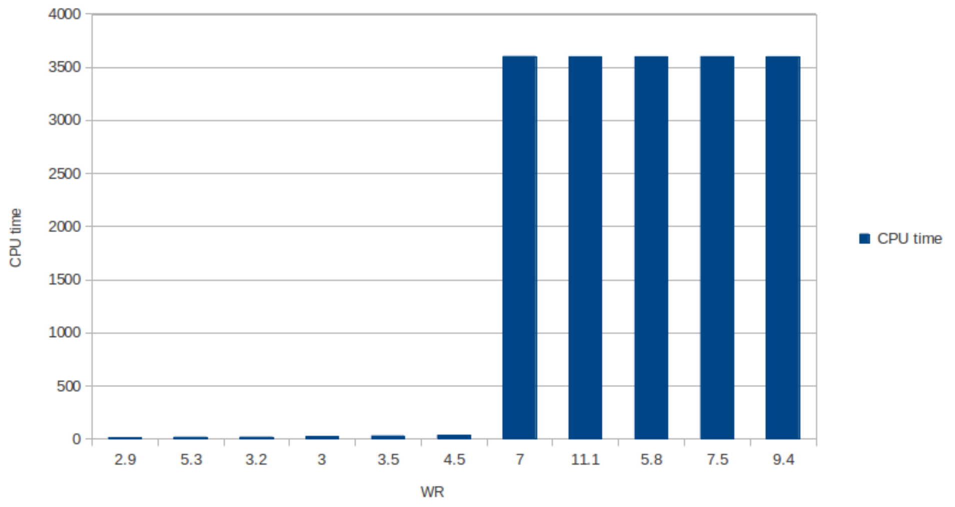


Figure 3: CPU time vs WR



All experiments are conducted on a machine with processor AMD Turion 64X2 Mobile Technology TL-50 (1.6 GHz, 2x 256 KB L2 cache) and 3.6 GiB memory running Ubuntu 12.04 as operating system.

4 Conclusion

Sensitivity of SALBP difficulty to precedence network is investigated in this study. Three measures that estimates difficulty using precedence network is considered. A Python class is created to conduct experiments.

Experimental results suggest that FR and WR predict the difficulty of the problems. Problem is expected to increase as FR and WR increases. Measures are good to use for relative comparison but may not always yield accurate results when interpretations depend on their magnitude. This means that if one have two problems with different FR/WR we expect the higher to be more difficult. If the ratio of the higher one is double of the lower one expecting the problem difficulty to be double (in terms of CPU time) is not an accurate prediction.

This study includes only a subset of the benchmark problems. There are also other measurements that predicts problem difficulty such as precedence index, task time index and project index. Using Salbp solver this study can easily be extended to include other problems and difficulty measures.

References

[1] İ. Baybars. A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 32(8):909–932, August 1986.

[2] E. M. Dar-El. Malb-a heuristic technique for balancing large single-model assembly lines. *AIIE Transactions*, 5:343–356, 1973.

[3] A. A. Mastor. An experimental investigation and comparative evaluation of production line balancing techniques. *Management Science*, 16:728–746, 1970.

[4] M. E. Salveson. The assembly line balancing problem. *Journal of Industrial Engineering*, 6:18–25, 1955.