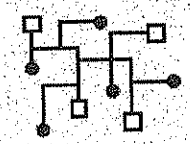

ISE

Industrial and Systems Engineering

Balancing U-Shaped Assembly Lines

Sihua Chen
Binney & Smith, Inc.

Louis Plebani
Lehigh University

Report No. 03T-007

LEHIGH
University

200 West Packer Avenue
Bethlehem, PA 18015

Balancing U-Shaped Assembly Lines

Sihua Chen
Binney & Smith, Inc.

Louis Plebani
Lehigh University

Report No. 03T-007

Balancing U-Shaped Assembly Lines

Sihua Chen[1] and Louis Plebani[2,*]

[1] Manufacturing Development Manager, Binney & Smith, Inc., 1100 Church Lane, P.O. Box 431, Easton, Pennsylvania. 18044-0431

[2]Department of Industrial and Systems Engineering, Harold S. Mohler Laboratory, 200 West Packer Avenue, Bethlehem, PA 18015-1582

Abstract

This paper presents a new heuristic and an optimal best first branch and bound procedure for the U-Shaped assembly line balancing problem. Results using standard assembly line balancing problem benchmarks show that the new heuristic provides improved solutions over previous heuristics without excessive increase in execution time. The new heuristic together with several lower bound heuristics adapted from bin packing and vehicle routing, are used to provide tight bounds to the best first branch and bound algorithm. Results are presented which shown the new branch and bound algorithm has excellent performance when compared to current algorithms.

1 Introduction

The assembly process consists of a precedence constrained sequence of tasks typically depicted as an acyclic Activity on Node (AON) network precedence diagram such as Figure 1 [32]. The nodes $\{k : t_k\}$ represent a task k that consumes deterministic time t_k and cannot be sub-divided. Arc (i, j) means that task i must be finished before task j can start.

The assembly line balancing problem (ALBP) is to group tasks into workstations such that the sum of the processing times at each station does not exceed a station uniform cycle time. There are two common versions of this problem: (ALBP-1) to find the smallest number of workstations for a given cycle time ; and (ALBP-2) to find the smallest cycle time for a given number of workstations. Virtually all researchers use search procedures for ALBP-2 which iteratively solve ALBP-1 problems obtained by varying the cycle time. This paper limits discussion to the solution of ALBP-1.

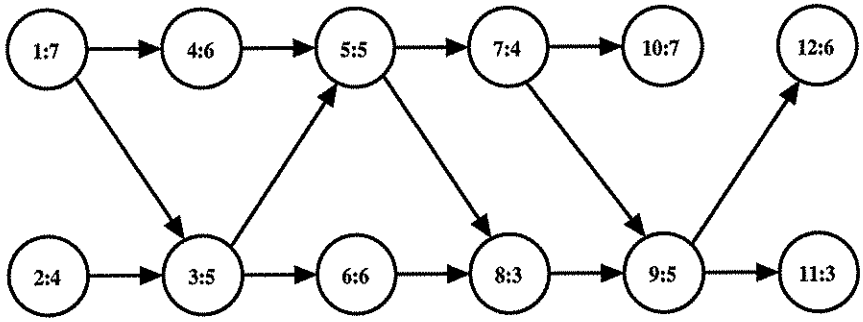

Figure 1: Example of A Precedence Network

Straight assembly line balancing (SALBP) has been studied for nearly five decades since first introduced by Salveson [29] and is known to be NP-Hard [15]. Heuristic methods such as genetic algorithms [5], filtered bean search [18], revised Hoffmann heuristic [7] and tabu search [25] are among the recent attempts to solve this problem. Integer Programming [4, 26, 34, 36], Dynamic Programming [10, 14, 30], and Branch and Bound approaches [9, 11, 13, 24, 27, 28, 31, 37] have been used for exact solution. Among these exact algorithms, branch and bound has consistently proven to be the best approach [2]. Figure 2 shows the optimal solution to the straight line problem

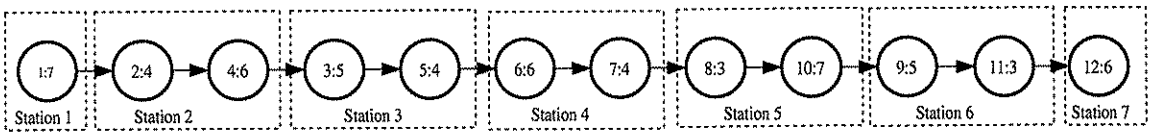

Figure 2: Straight Assembly Line Solution to Figure 1

(SALBP) corresponding to Figure 1 and a cycle time of 10. It can be seen that seven workstations are needed, each grouping the tasks as shown.

Largely due to pressures of JIT, many assembly lines are now designed as U-shaped assembly lines because of their potential for better balancing, improved visibility and communications, fewer work stations, more flexibility for adjustment, minimization of operation travel, and easier material handling when compared to straight lines [19]. Figure 3 shows an optimal U-shape assembly line for the example network of Figure 1. Only six workstations are required as compared to the seven required in the straight assembly line case.

U-shaped line balancing is a relative new research topic having first been introduced and formulated by Miltenburg and Wijngaard in 1994 [19]. Since then, there has been a modest amount of heuristics work [1, 6, 8, 12, 19, 21–23, 32] and few ex-

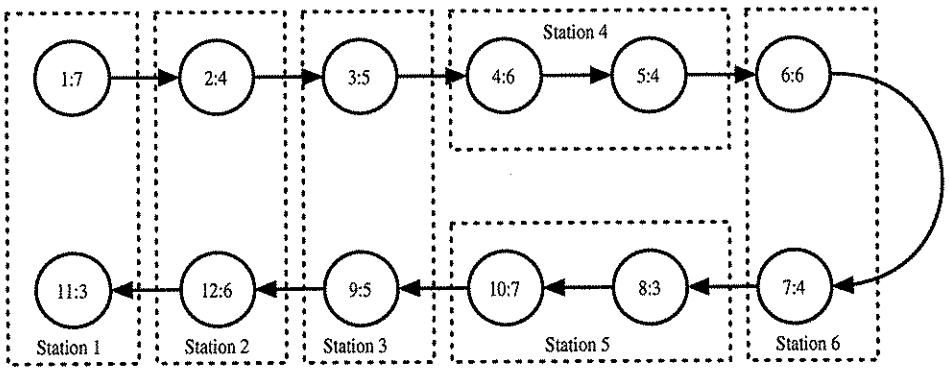

Figure 3: U-shaped Assembly Line Solution to Figure 1

act solution approaches to the problem. Sparling introduced depth first branch and bound, best first branch and bound, and dynamic programming for U-shaped lines of up to 30 tasks in 1997 [33]. Scholl and Klein created a depth-first station-oriented branch and bound algorithm called ULINO, which was a clone of their straight line algorithm called SALOME [32], in 1999.

This paper reports a best-first branch and bound algorithm called OBUL (Optimal Balancing of U-shaped Lines). Motivations for a depth first B&B approach were two-fold: (1) Baybars concluded that branch and bound is the best optimal solution methodology for straight assembly line balancing [2]; (2) Sparling's research for U-shaped assembly line balancing, in particular, and Morton's research, in general, has shown that a best-first branch and bound approach is superior to depth-first methods if high quality lower and upper bounds are available [20]. OBUL uses multiple lower bound algorithms and a new upper bound heuristic to achieve tighter bounding.

2 Network representation

In addition to the information contained in the precedence network, other attributes are calculated for convenience in algorithm processing. The resulting representation of the network $\mathcal{N}$ is a list of node objects where each node contains the following information ($x \prec y$ indicates x is constrained to precede y in the assembly sequence):

n : node number.

t_n : Processing time of task n.

$P1_n$ = $\{i \mid (i,n) \in \{(i,j)\}\}$. The set of immediate predecessors of node n.

$S1_n$ = $\{j \mid (n,j) \in \{(i,j)\}\}$ The set of immediate successors of node n.

$P2_n$ = $\bigcup_{k \prec n} P1_k$. The set of all predecessors of node n not contained in $P1_n$.

$S2_n$ = $\bigcup_{k \succ n} S1_k$. The set of all successors of node n not contained in $S1_n$.

3 Bounds

OBUL achieves reasonably tight bounds through the use of multiple lower bound algorithms and a new heuristic called Immediate Update Score First Fit (IUSFF).

3.1 Lower bound

OBUL uses four lower bound algorithms. In addition to the traditional lower bound $LB_1 = \sum_i t_i/n$, OBUL uses three bounds based upon arguments from bin packing, machine scheduling and vehicle routing. LB_2 and LB_3 can prove useful when the cycle time is small enough to result in three or fewer tasks per station, which is where strong bounds are most needed [13]. LB_2 is the number of tasks which have a duration greater than half the cycle time, since these cannot share a station, plus one half the number of tasks equal to exactly half a cycle time.

$$LB_2 = N\left(C/2, C\right] + \left\lceil \frac{1}{2} N\left[C/2\right] \right\rceil \tag{1}$$

LB_3 partitions the task times into sets of third's of cycle time $C/3$. Any task with task time greater than $2C/3$ will occupy a single station, tasks with task time exactly equal to $2C/3$ may share a station with a task whose task time is exactly $C/3$ or less. No more than two tasks with task time between $C/3$ and $2C/3$ may share a station with each other.

$$LB_3 = \left\lceil N\left(2C/3, C\right] + \frac{2}{3} N\left[2C/3\right] + \frac{1}{3} N\left[C/3\right] + \frac{1}{2} N(C/3, 2C/3) \right\rceil \tag{2}$$

LB_4 is a lower bound developed by Labbé [16] in a vehicle routing context and was found by Berger [3] to be a strong lower bound for assembly line balancing. The

bound is calculated in several steps:

Step 1: All tasks are sorted in nondecreasing order of processing times. $W(a,b]$ is defined as the set of tasks with $a < t \leq b$. Starting with the longest task, assign the tasks of $W(C/2,C]$ to different stations, since any two of them have a total processing time exceeding C.

Step 2: Consider the tasks of $W(C/3,C/2]$. Starting with the smallest task, and all stations corresponding to the tasks of $W(C/2,C]$, in non-decreasing order of idle times. Successively assign the task of $W(C/3,C/2]$ to the first available station. Stop when all these tasks have been assigned or no station remains.

Step 3: Let K be the number of unassigned tasks of $W(C/2,C)$ after step 2. Then at least $\lceil K/2 \rceil$ are required since at most two tasks of $W(C/3,C/2)$ can be assigned to the same station.

Step 4: Determine the number of additional stations required for all unassigned tasks. Based upon arguments by Martello and Toth [17], the number of additional stations required for all tasks for which $t > c$ is:

$$p(c) = \max\left\{0, \left\lceil \frac{\sum_{i \in W(c,C-c]} t_i - (|W(C/2,C-c)|) + \lceil K/2 \rceil)\, C}{C} \right\rceil \right\} \qquad (3)$$

This calculation of $p(c)$ is based on the fact that a task with $t > c$ cannot be executed on a station which is already busy for at least $C - c$ time units. Since equation 3 holds for any value of c, and $p(c) = 0$ for $c \geq C/3$:

$$LB4 = \max_{0 \leq c < C/3} \{(|W(C/2,C-c)|) + \lceil K/2 \rceil) + p(c)\} \qquad (4)$$

3.2 IUSFF heuristic for upper bound

IUSFF (Immediate Update Score First Fit) is a modification of Hackman's IUFF (Immediate Update First Fit) heuristic for straight assembly line balancing [9] which in turn was based upon the GFF (Generalized First Fit) heuristic described by Wee and Magazine [38]. Similar to these heuristics, IUSFF balances the line by scoring, ranking, and assigning tasks to workstations until none remain. Four separate scoring functions are used: (WE) work element time, the processing time of a task; (PW)

positional weight, the sum of the processing times for a task and all the tasks that must follow (precede) it; (NF) number of followers, the number of tasks that follow (precede) a task; and (NIF) number of immediate followers, the number of tasks that immediately follow (precede) a task. Investigation determined that it was not possible to predict which scoring function would provide the best solution for any particular network. Thus the default procedure is to apply the complete algorithm using each of scoring functions separately and choose the best of the resulting four solutions.

In addition to obvious modifications necessary to accommodate U-shaped lines, IUSFF improves over earlier heuristics by immediately adjusting scores and re-ranking available tasks after each task assignment. The adjustment and re-ranking is critical in U-Shaped line assignments since tasks can be assigned from either end of the line which immediately affects the rankings scores in the case of PW, NIF, and NF criteria.

The IUSFF algorithm repeatedly creates a new station and fills it with tasks until no tasks remain to be assigned. Input is the precedence network $\mathcal{N}$ representation from section 2, the set of tasks that have already been assigned T, and the cycle time C. The following general heuristic is repeated until all tasks are assigned:

Step 1: Identify the set of unassigned tasks U:

$$U = \mathcal{N} - T \tag{5}$$

If $U = \emptyset$ the algorithm is terminated.

Step 2: Instantiate a new station. Major data members of the new station are:

$slack$: the slack for the current station. Initialized to C.

T_n : the set of tasks assigned to this (n^{th}) station. Initialized to $\emptyset$.

A : set of tasks which are available for assignment to the current station.

rs_k : ranking score of task k.

Step 3: A is initialized by performing

$$\forall\, k \in U \quad : \quad \begin{cases} \text{if } P1_k \cap T' = \emptyset & \text{execute step 4 and return} \\ \text{if } S1_k \cap T' = \emptyset & \text{execute step 5 and return} \end{cases} \tag{6}$$

After initialization go to step 6.

Step 4: Task k has been determined to have no unassigned predecessors. Update A and calculate the ranking score for a task that is predecessor free using the current scoring function:

$$A = A \cup \{k\} \tag{7}$$

$$\text{PW}: \quad rs_k = t_k + \sum_{i \in S1_k \cap U} t_i + \sum_{i \in S2_k \cap U} t_i \tag{8}$$

$$\text{WE}: \quad rs_k = t_k \tag{9}$$

$$\text{NF}: \quad rs_k = |S1_k \cap U \mid + \mid S2_k \cap U \mid \tag{10}$$

$$\text{NIF}: \quad rs_k = |S1_k \cap U \mid \tag{11}$$

Step 5: Task k has been determined to have no unassigned successors. Update A and calculate the ranking score for a task that is successor free using the current scoring function:

$$A = A \cup \{k\} \tag{12}$$

$$\text{PW}: \quad rs_k = t_k + \sum_{i \in P1_k \cap U} t_i + \sum_{i \in P2_k \cap U} t_i \tag{13}$$

$$\text{WE}: \quad rs_k = t_k \tag{14}$$

$$\text{NF}: \quad rs_k = |P1_k \cap U \mid + \mid P2_k \cap U \mid \tag{15}$$

$$\text{NIF}: \quad rs_k = |P1_k \cap U \mid \tag{16}$$

Step 6: If $A = \emptyset$, the station is fully loaded. Go to step 12.

Step 7: A is sorted ascending based on the following boolean condition which defines task x less than task node y:

$$x < y \equiv (rs_x < rs_y) \ \vee \ (rs_x = rs_y) \wedge (t_x < t_y) \tag{17}$$

Step 8: Determine the task with the highest ranking score which is feasible for the

current station using the following operations (A is processed as a LIFO stack):

$$\begin{aligned} &\text{while } A \neq \emptyset \ : \\ &\qquad k = A.\text{Pop} \\ &\qquad \text{if } t_k \leq slack \text{ then go to step 9} \end{aligned} \tag{18}$$

If $A = \emptyset$, the station is fully loaded. Go to step 12

Step 9: Task k (determined in step 8) is assigned to the currently opened station.

$$slack = slack - t_k \tag{19}$$

$$T_n = T_n \cup \{k\}\,, \tag{20}$$

$$T = T \cup \{k\} \tag{21}$$

If $slack = 0$, the station is fully loaded. Go to step 12.

Step 10: Ranking scores for the tasks which remaining in A are immediately adjusted for the effects of adding task k if the ranking criteria is PW, NF, or NIF. No adjustment is necessary for the WE ranking criteria. The adjustments are made by iterating through the nodes in A and making adjustments in the ranking score for the effect of adding task k as appropriate for the scoring criteria in effect.

$$\forall\, k \in A \quad : \quad \begin{cases} \text{PW}: & \text{if } (k \in P2_k \ \vee \ k \in S2_k), \text{ then } rs_k = rs_k - t_k \\ \text{NF}: & \text{if } (k \in sP2_k \ \vee \ k \in sS2_k), \text{ then } rs_k = rs_k - 1 \\ \text{NIF}: & \text{if } (k \in sP1_k \ \vee \ k \in sS1_k), \text{ then } rs_k = rs_k - 1 \end{cases} \tag{22}$$

Step 11: Check availability criteria for all predecessors and successors of task k just assigned.

$$\begin{aligned} &\forall\, i \in S1_k : \text{if } P1_i \cap T' = \emptyset \\ &\qquad \text{calculate } rs_i \text{ as in step 4} \\ &\qquad A.\text{Push}(k). \end{aligned} \tag{23}$$

$$\begin{aligned} &\forall\, i \in P1_k : \text{if } S1_i \cap T' = \emptyset \\ &\qquad \text{calculate } rs_i \text{ as in step 5} \\ &\qquad A.\text{Push}(k). \end{aligned} \tag{24}$$

Go to step 6

Step 12: At this point the current open station is completely loaded based upon the tasks that are currently available. i.e., there is no predecessor or successor free task with time is less than the current slack time of the open station. Go to Step 1.

3.3 Performance of the IUSFF heuristic

The IUSFF heuristic and the IUFF heuristic were each used to solve the data sets of Talbot and the data sets of Hoffmann. These data sets have been established as standard benchmarks in the literature [11, 35]. The Talbot data set is based on 12 precedence networks with 8-111 tasks combined with several cycle times for a total of 64 instances. The Hoffman data set uses 30-111 tasks for a total of 50 instances. Both algorithms were implemented in C++ and run on a Pentium III laptop. Results are summarized in Table 1 where relative deviation is defined as $(UB - LB)/LB$. UB is the value reported by the respective algorithm, and LB is either the optimal solution or the best known lower bound to the associated problem. It can be seen that IUSFF provides superior results without excessive processing time. The average deviations for IUSFF are approximately 28 to 55 percent smaller than those of IUFF. IUSFF found 47 optimal solutions versus 34 for IUFF for the Talbot data set and 15 optimal solutions for the Hoffmann data set versus 6 for IUFF.

Table 1: IUSFF/IUFF Comparison

	IUSFF		IUFF	
	Talbot	Hoffmann	Talbot	Hoffmann
Number of Problems	64	50	64	50
Optimal Solutions Found	47	15	34	6
Average Relative Deviation	0.039	0.056	0.088	0.078
Max Relative Deviation	0.33	0.20	0.50	0.20
Average cpu secs	0.013	0.034	0.009	0.017

4 OBUL Branch and Bound

OBUL is a station oriented best first search branch and bound algorithm. The major data structure of OBUL is a priority queue of branch and bound nodes (BBN's). Each

BBN represents a partial solution. A BBN representing stations $1 \ldots n$ is at level n in the B&B tree. Major data members of the BBN are:

T : the set of all tasks assigned to all stations up to and including current station being filled,

slack : the slack time at the current station,

ts : total slack time for all stations up to and including the current station.

In the priority queue, BBN node x is ranked higher than BBN node y, i.e., more desirable for branching, if the following boolean is true:

$$\begin{aligned}(x.LB < y.LB) \;\vee\; (x.LB = y.LB) \wedge (x.ts > y.ts) \\ \vee\; (x.LB = y.LB) \wedge (x.ts = y.ts) \wedge (x.n > y.n)\end{aligned} \tag{25}$$

Nodes are continually popped from the priority queue and processed until the queue is empty, or the global upper bound becomes equal to the global lower bound, or an optional user specified time limit is reached. As each node is popped, fathoming tests are performed and the branching algorithm is applied. The branching algorithm pushes new feasible partial solutions onto the priority queue. If execution time exceeds the optional user specified limit, the B&B terminates and a heuristic finishing procedure is used.

4.1 Branching Algorithm

Input to the branching algorithm is the newly popped BBN representing the tasks that have been assigned to n stations. The purpose of the branching algorithm is to create new BBN's by adding all possible feasible sets of tasks (which form new filled stations), to the input node and to push each newly created node onto the priority queue after passing a series of fathoming tests. The new nodes are, by definition, at depth $n + 1$ of the branch and bound tree. Much of the processing time of the branching algorithm is consumed by determining all sets of feasible tasks that could fully populate a new station. Key working storage variables are:

avail : list of tasks which are available for assignment to the present station.

unav : a set of which are forbidden to be assigned to the present station.

sts : the smallest task time of those available tasks which were specifically omitted from the immediate assignment sequence.

The major steps of the branching algorithm are:

Step 1: Initialize working storage from the attributes of the input BBN node:

$$unav = T, \quad Avail = \emptyset, \quad slack = 0, \quad sts = \infty \tag{26}$$

$$\forall\ k \in \mathcal{N}\ :$$

$$\text{if } (k \notin unav)\ \wedge\ (P1_k \cap T' = \emptyset) \vee (S1_k \cap T' = \emptyset)\ :\ Avail.\text{Push}(k) \tag{27}$$

Step 2: This step is the first of a recursive sequence. In addition to the initial entry from the outer loop, it will be entered numerous times as the algorithm proceeds. *avail* is sequentially searched for a task $k \notin unav$ that can "fit" into the current station, i.e., with processing time $t_k \leq slack$.

If a task is found, the task is assigned to the current station by executing step 3. Eventually there is a return to this step at this point. Upon return, the current task k is removed from the assignment by updating bookkeeping variables:

$$slack = slack + t_k, \quad sts = \min(sts, t_k) \tag{28}$$

and by removing all tasks from *avail* which the recursive processing has added since task k was assigned. The search resumes starting with the task immediately after task k.

When the end of *avail* is reached, if the current recursive depth is at the outer level, branching on the input BBN terminates, otherwise go to step 5.

Step 3: Task k is assigned to the current station by updating node and bookkeeping variables:

$$T = T \cup k, \quad slack = slack - t_k, \quad unav = unav \cup \{k\} \tag{29}$$

If $slack = 0$, go to step 5 (station has been filled), otherwise continue with step 4.

Step 4: Check for tasks that become available due to task k being assigned to the present station. The only tasks that could become available are the immediate

predecessors and the immediate successors task k. Tasks that become available are added to *avail*.

$$\forall\, p \in P1_k \ :$$
$$\text{if } (p \notin unav) \wedge (S1_p \cap T' = \emptyset) \ : \ Avail.\text{Push}(p),\ unav = unav \cup p \qquad (30)$$
$$\forall\, s \in S1_k \ :$$
$$\text{if } (s \notin unav) \wedge (P1_p \cap T' = \emptyset), \ : \ Avail.\text{Push}(s),\ unav = unav \cup s \qquad (31)$$

Go to step 2 which starts a new recursive search for next task to assign.

Step 5: A feasible assignment of tasks has been found. The following series of fathoming tests is performed in the order listed:

1. If all tasks have been assigned go to step 6,
2. The maximum load rule is applied. If during the process of creating the immediate assignment, a task was skipped that could fit into the current assignment ($sts \leq slack$), discard the current assignment and return to the appropriate recursive level in step 2.
3. If a solution is to be better than the current global upper bound (GUB) then
$$ts \leq (GUB - 1) * C - \sum_{\forall k} t_k \qquad (32)$$
must be true, otherwise, discard the current assignment and return to the appropriate recursive level in step 2.
4. The priority queue is checked for a `BBN` that is both at the same depth as the candidate node and has the identical total assignment T as the candidate node. If such a node is exists, discard the current assignment and return to the appropriate recursive level in step 2.
5. The lower bound algorithms are applied to the tasks remaining to be assigned ($\mathcal{N} - T$). If the returned LB plus the current number of stations is greater than or equal to GUB, discard the current assignment and return to the appropriate recursive level in step 2.

If all fathoming tests are passed, push the candidate `BBN` onto the priority queue, and return to the appropriate recursive level in step 2.

Step 6: A complete feasible solution has been found. The solution is used to update the global upper bound. If GUB becomes equal to the current global lower bound of the problem, the optimal problem assignment has been found. Otherwise, return to the appropriate recursive level in step 2.

4.2 Heuristic Finishing Procedure

This procedure is executed if the branch and bound portion of OBUL exceeds the user specified allowable execution time limit. This ensures that the algorithm always returns at least one feasible solution. The procedure randomly selects a user specified parameter maximum (we used 100) of BBN's from those that remain in the priority queue. For each BBN, the number of stations n is combined with the result of the IUSFF algorithm is applied to the remaining $\mathcal{N} - T$ tasks. The best solution is reported.

5 Computational Results and Discussion

5.1 Computational Results

OBUL was coded using C++ and run on a Pentium III laptop computer. The computational results are compared with the reported results of the best known algorithm, Scholl and Klein's depth first branch and bound algorithm ULINO. The Talbot and Hoffmann's data sets are used for comparison. OBUL's user specified time limit was set to 500 seconds. Overall results are shown in Table 2. In the Talbot data set OBUL found 3 more optimal solutions than ULINO. In the Hoffmann data set, OBUL found 5 more optimal solutions than ULINO.

Table 2: OBOL/ULINO Comparison

	OBUL		ULINO	
	Talbot	Hoffmann	Talbot	Hoffmann
Number of Problems	64	50	64	50
Optimal Solutions Found	63	37	60	32
Average Relative Deviation	0.003	0.019	0.004	0.020
Max Relative Deviation	0.071	0.091	0.071	0.100

The influences of network structure were studied by considering three major parameters of network structure: the number of tasks (NT); the order of strength (OS);

and time variability ratio (TV).

5.1.1 The Number of Tasks (NT)

The combined Talbot and Hoffmann data set was partitioned into small, medium, and large groups as shown in Table 3. With increasing NT, the number of optimal

Table 3: Influence of Number of Tasks (NT)

NT Class	# Instance	# Opt	Ave. Dev.	Ave.Time (s)
$NT \leq 25$	27	100 %	0	0.02
$25 < NT \leq 80$	49	93.9 %	0.005	58.12
$NT > 80$	38	63.2 %	0.014	179.11

solutions decreased from 100% to 63.2%, and average calculation time increased from 0.0011 second to 179.11 seconds. This is primarily due to the branch and bound tree growing with the number of tasks.

5.1.2 The Order of Strength (OS)

The order of strength is defined by the number of arcs in the precedence network divided by $n(n-1)/2$. It measures the relative number of precedence relations in the precedence network. In the Talbot and Hoffmann combined data set, the minimum OS is 22.5% and the maximum is 83.3%. This range was partitioned into small, medium, and large groups as shown in Table 4. With increasing OS, the percentage

Table 4: Influence of Order of Strength (OS)

OS Class	# Instance	# Opt	Ave. Dev.	Ave.Time (s)
$OS \leq 45\%$	52	73.07 %	0.0102	130.37
$45\% < OS \leq 65\%$	50	94 %	0.0044	48.5
$OS > 65\%$	12	100 %	0	0.005

of optimal solutions improved, and average calculation time decreased from 130.37 seconds to 0.005 second. This mainly due to the fact that order of strength limits

the number the number of feasible task combinations which results in the branch and bound tree being reduced in size.

5.1.3 The time Variability Ratio (TV)

The time variability ratio is defined as $TV = \frac{t_{max}}{t_{min}}$. It measures the time structure in the precedence network. In the Talbot and Hoffmann combined data set, the minimum TV is 5.7 and the maximum is 568.9. This range was partitioned into small, medium, and large groups as shown in Table 5. With increasing TV, the percentage

Table 5: Influence of Time Variability (TV)

TV Class	# Instance	# Opt	Ave. Dev.	Ave.Time (s)
$TV \leq 15$	24	100 %	0	0
$15 < TV \leq 80$	47	100 %	0	9.255
$TV > 80$	12	67.44 %	0.0349	203.94

of optimal solution decreased, and average calculation time increased from 0 second to 204 seconds. A primary explanation of this is that large task times require large task time. If relatively smaller task times exist then the number of feasible combinations of tasks to fill a station gets large which increases the number of branching computations at each branch and bound node.

6 Summary

The U-shaped assembly line balancing problem has been addressed in this paper. A new heuristic algorithm IUSFF was introduced that improved upon solution quality of earlier heuristics without excess computation time requirements. A best first branch and bound algorithm call OBUL was presented which used IUSFF to obtain initial upper bounds. Results showed that with improved bounds, the performance of a best first branch and bound can compete with that of the best known depth first algorithm.

References

[1] D. Ajenblit and R. Wainwright. Applying genetic algorithms to the u-shaped assembly line balancing problem. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 96–101, 1998.

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

[3] Bourjollly J. Berger, I. and G. Laorte. Branch-and-bound algorithms for the multi-product assembly line balancing problem. *European Journal of Operations Research*, 58:215–222, 1992.

[4] E. Bowman. Assembly-line balancing by linear programming. *Operations Research*, pages 385–389, 1960.

[5] B.A.; Redfern M.S. Carnahan, B.J.; Norman. Incorporating physical demand criteria into assembly line balancing. *IIE Transactions*, 33:875–887, 2001.

[6] P. Chiang, W. Kouvelis and C. Chen. An efficient heuristic for the u-shaped assembly line balancing problem in the just-in-time production environment. In *Proceesings of National Annual Meeting to the Decision Sciences*, page 1126, 1997.

[7] K.S. Fleszar, K.; Hindi. An enumerative heuristic and reduction methods for the assembly line balancing problem. *European Journal of Operational Research*, 145:606–620, 2003.

[8] F Guerriero and J. Miltenburg. The stochastic u-line balancing problem. *J. Naval Research Logistics*, 50(1):31–57, 2003.

[9] S. Hackman, M. Magazine, and T. Wee. Fast, effective algorithms for simple assembly line balancing problems. *Operations Research*, 37(6):916–924, 1989.

[10] R. Held, M. Karp and R. Shareshian. Assembly-line balancing–dynamic programming with precedence constraints. *Operations Research*, pages 442–459, 1963.

[11] T. Hoffmann. Eureka: a hybrid system for assembly line balancing. *Management Science*, 38(1):39–47, 1992.

[12] H. Hwang, J. U. Sun, and T. Yoon. U-line line balancing with simulated annealing. In *Proceedings of the First Asia-Pacific decision sciences institute conference*, pages 101–108, Hong Kong, June 1996.

[13] R. Johnson. Assembly line balancing: a branch and bound algorithm and computational comparison. *International Journal of Production Research*, 19(3):277–287, 1981.

[14] E. Kao and M. Queyranne. On dynamic programming methods for assembly line balancing. *Operations Research*, 30:375–390, 1982.

[15] R. M. Karp. Reducibility among combinatorial problems. *In Complexity of Computer Computations, (Proc. Sympos. IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y.).New York: Plenum,*, pages 85–103, 1972.

[16] Laporte G. Labbé, M. and H. Mercure. Capacitated vehicle routing on trees. *Operations Research*, 39:616–622, 1991.

[17] S. Martello and P. Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete Applied Mathematics*, 28:59–70, 1990.

[18] C.A. Matanachai, S.; Yano. Balancing mixed-model assembly lines to reduce work overload. *IIE Transactions*, 33:29–42, 2001.

[19] G. J. Miltenburg and J. Wijngaard. The u-line balancing problem. *Management Science*, 40(10):1378–1388, 1994.

[20] T. Morton and D. Pentico. *Heuristics scheduling systems.* John Wiley and Sons, Inc, 1993.

[21] K. Nakade and K. Ohno. Analysis and optimization of a u-shaped production line. *Journal of Production Research Society of Japan*, 40(1):90–104, 1996.

[22] K. Nakade and K. Ohno. Stochastic analysis of a u-shaped production line with multiple workers. *Computers and Industrial Engineering*, 33(3-4):809–812, 1997.

[23] K. Nakade and K. Ohno. An optimal worker allocation problem for a u-shaped production line. *International Journal of Production Economics*, 60-61(3-4):353–358, 1999.

[24] F. Nourie and E. Venta. Finding optimal line balances with optpack. *Operation Research Letters*, 10:165–171, 1991.

[25] C.; Duran A.; Perez M. Pastor, R.; Andres. Tabu search algorithms for an industrial multi-product and multi-objective assembly line balancing problem, with reduction of the task dispersion. *Journal of the Operational Research Society*, 53:1317, 2002.

[26] J. Patterson and J. Albracht. Assembly-line balancing: zero-one programming with fibonacci search. *Operation Research*, pages 166–172, 1973.

[27] D. Pinto, P. Dannexbring and M. Khumawala. A branch and bound and algorithm for assembly line balancing with paralleling. *International Journal of Production Research*, 13(2):183–196, 1975.

[28] D. Pinto, P. Dannexbring and M. Khumawala. Branch and bound and heuristic procedures for assembly line balancing with paralleling of stations. *International Journal of Production Research*, 19:565–576, 1981.

[29] J. H. Salveson. The assembly line balancing problem. *Journal of Industrial Engineering*, 6(3):18–25, 1955.

[30] E. Sarin, S. Erel and E. Dar-El. A methodology for solving single model stochastic assembly line balancing problem. *Omega, the internatioanl Journal of Management Science*, 27:525–535, 1999.

[31] A. Scholl and R. Klein. Salome, a bidirectional branch-and-bound procedure for assembly line balancing. *INFORMS Journal on Computing*, 9(4):319–334, 1997.

[32] A. Scholl and R. Klein. Ulino: Optimally balancing u-shaped jit assembly lines. *International Journal of Production Research*, 37:721–736, 1999.

[33] D. Sparling. *Topics in U-line Balancing.* PhD thesis, McMaster University, Canada, 1997.

[34] F. Talbot. An integer programming algorithm with network cuts for solving the assembly line balancing problem. *Management Science*, 30:85–99, 1984.

[35] J. Talbot, F. Patterson and W. Gehrlein. A comparative evaluation of heuristic line balancing techniques. *Management Science*, 32(4):430–454, 1986.

[36] T. Urban. Note: Optimal balancing of u-shaped assembly lines. *Management Science*, 44:738–741, 1998.

[37] F. Van Assche and W. Herroelen. An optimal procedure for the single-model deterministic assembly line balancing problem. *European Journal of Operations Research*, 3:142–149, 1978.

[38] T. Wee and M. Magazine. Assembly line balancing as generalized bin packing. *Operations Research Letters*, 1:56–58, 1982.