Heuristic for Task-Worker Assignment with Varying Learning Slopes

Kanjana Thongsanit, Wipawee Tharmmaphornphilas and Rein Boondisakulchok*

Department of Industrial Engineering, Faculty of Engineering, Chulalongkorn University, Bangkok, Thailand 10330
E-mail : kanjanath7@yahoo.com, wipawee.t@chula.ac.th and rein.b@chula.ac.th*

ABSTRACT

When a worker performs a task repetitively, s/he requires less time to produce the succeeding units of a task due to his/her learning ability. In mass production, a constant production rate assumption is always assumed in developing a task-worker assignment since the learning period is only a small part compared to the overall production period. However, in the fashion industry, new product styles are launched more frequently and order sizes are smaller. Due to this small lot size, task-worker assignments based on a constant production rate assumption may not be applicable. As a result, learning should be considered in the assignments in the fashion industry. This paper proposes a method of task-worker assignment considering worker skill levels and learning abilities. The processing time of each worker varies in the production period depending on worker learning ability. We focused on task-worker assignments where tasks are ordered in a series and the number of tasks is greater than the number of workers. Workers could perform multiple tasks with the precedence restriction. An integer linear programming model was formulated with the objective to minimize makespan. A heuristic was proposed to find the best assignment. The performance of the heuristic method was tested by comparing the quality of solution and computational time to optimal solutions. The experimental results show that the heuristic provides good solutions with a better CPU time compared to the optimal solutions. For problems 7, 8 and 9 workers, the heuristic found a solution within 3% of the optimal solutions with 0.67% on average.

KEYWORDS

heuristics, assembly line balancing, assignment problem, skill and learning

doi:10.4186/ej.2010.14.2.1

I . Introduction

In the fashion industry, new product styles are launched more frequently than in the past with smaller lot sizes due to increased market competition. Consequently, a trend in garment production in Thailand has emerged shifting from mass production to small lot production since it is more flexible and responsive compared to mass production. Garment production is labor intensive and the production rate mainly depends on worker skills. With experience and training, each worker has different skill levels and learning ability.

Generally, people learn and improve their performance by repeating operations. They will require less time to produce the succeeding unit or gain proficiency with the repetition of the same task. This is called learning behavior. Learning behavior is examined and presented in mathematical models [1,2]. Many factors influence learning e.g. job complexity, the number of repetitions, previous experience and training [2]. Many studies related to production address learning e.g. scheduling [3], assembly line balancing [4], allocation or worker selection [5].

In mass production, learning behavior is usually not considered. A constant production rate assumption is always assumed in developing a task-worker assignment since the learning period is only a small part compared to a whole production period. However, in the fashion industry, new product styles are launched more frequently and lot sizes are smaller. With this small lot size, the learning period becomes a more substantial part of production time. A task-worker assignment with a constant production rate assumption may not directly apply since it may not provide the optimal solution in practice. For this reason, learning should be considered in a task-worker assignment in the fashion industry.

A conventional assembly line balancing problem is one of the workstation assignment problems which concern the task precedence constraint. It is designed for mass production and the processing time of every item in order is assumed equivalent. However, for small lot production, learning cannot be ignored [6]. The objective of addressing this problem is minimizing the maximum workstation time or cycle time. However, this cannot be applied when learning is considered since the bottleneck dynamically changes based on the reduction of learning slope of each worker. Therefore, considering learning minimizes makespan in regards to this problem.

An assembly line balancing problem with learning consideration is studied in different assumption. Toksari, [7] assumed that the learning for all tasks is same. Chakravarty,[4] , Karni and Herer [6], Dar-El [8] and Cohen, [9] defined that learning depends on the task which is assigned. Cohen [10] assumed learning depends on the worker who operated the task. Furthermore, the processing time which represents learning behavior is set in different ways. For example Karni and Herer[6] ; Chakravarty[4] set processing time in discrete whereas Cohen and Dar-El [11] , Cohen, [9] and Cohen, [10] used continuous which is represented by a learning model. Moreover Cohen, [9] and Cohen,[10] assumed that tasks can be divisible whereas Karni and Herer, [6], Chakravarty,[4] and Cohen and Dar-El [12] assumed the tasks cannot be split.

Most previous studies on the assembly line balancing problem considering learning ability assumed no buffer space between workstations. That is a paced line where accessible time to complete tasks at each workstation is limited by a maximum workstation time or bottleneck. When the bottleneck station finishes its tasks, all items between stations are simultaneously transferred [13]. However, when learning is considered, the bottleneck station dynamically changes place. The inter-departure time of finished items depends on the bottleneck time, so the sum of the bottleneck time becomes the completion time of the last item or makespan. Cohen [9] developed the upper envelope, which is the line that is formed by the maximum workstation time or bottleneck time, and showed that the area under the envelope will be the makespan. Therefore, minimizing the area under the envelope is minimizing the makespan. Cohen [9], Cohen [10] and Cohen and Dar-El [11] developed a heuristic and method to determine the optimal assignment based on the upper envelope concept.

This study, however, is concerned with the problem where buffer spaces between work stations are allowed. The buffer space is used to hold items when the next station is still working on the previous item. Buffers allow workstations to operate more independently when the production rate is different. Buffers are also introduced as decoupling agents to smooth and balance the flow of items between successive stations [14]. We focused on task-worker assignments in the assembly line balancing problem in the fashion industry. Workers have different abilities in learning and tasks are set in series. Multiple assignments are only allowed for consecutive tasks. To complete production, all tasks must be done and all workers must be utilized. In this paper, a heuristic for worker-task assignment based on the consideration of learning in order to minimize makespan was developed.

II. Problem Statement

This problem concerns assembly lines which contain a set of sequential workstations. Buffer spaces are set up between workstations. A workstation consists of a worker who carries out assigned tasks. An order includes *i,* which are identical items that must be processed along the same route passing through all workstations. This problem assumes that the workers have multiple skills and are able to do more than one task. The skill levels of the workers differ; therefore, workers' processing times vary.

Moreover, workers have different learning abilities i.e. the processing time of the succeeding item being shorter than the preceding item. In the problem, the processing time of each worker for each item is based on the skill level and learning of the worker. Table 1 provides an example of processing times applied in the problem.

Table 1
The processing times of 5 tasks, 3 workers (A, B, C) and 3 items

Worker A	**items 1**	**items 2**	**items 3**
Task 1	4	4	3
Task 2	3	2	1
Task 3	7	4	4
Task 4	2	1	1
Task 5	4	4	3

Worker B	**items 1**	**items 2**	**items 3**
Task 1	5	4	3
Task 2	3	3	2
Task 3	6	6	4
Task 4	3	1	1
Task 5	5	4	2

Worker C	**items 1**	**items 2**	**items 3**
Task 1	4	2	2
Task 2	4	2	2
Task 3	7	6	5
Task 4	2	2	1
Task 5	6	2	2

We want to establish worker assignments where all tasks must be assigned to workers. Each worker is assigned to at least one task. Since it is assumed that the number of tasks is greater than the number of workers, some workers can perform multiple tasks. Only consecutive tasks are allowed in the multiple assignments to smooth out the line. If a worker is assigned to more than one task, the worker processing time is determined by the sum of the processing time of all tasks that s/he performs. For example, if task 1 and task 2 are assigned to worker *A*, then the task processing times are 7, 6, 4 for items 1, 2, and 3 respectively. Tasks cannot be split. This problem is also based on the following assumptions. 1) The other learning factors among the tasks at the same workstation are not considered such as the task similarity. 2) There is unlimited buffer space between each workstation. The objective is to minimize the makespan, which is the completion time of the last item of the last task. The problem is formulated in an integer linear programming model.

The mathematical model presented below determines which tasks $\{j = 1,\ldots,o\}$ and workers $\{w = 1,\ldots,k\}$ are assigned to the workstations $\{s = 1,\ldots,k\}$. Workers must complete all items $\{m = 1,\ldots,i\}$ with the minimum completion time. The model uses three types of continuous decision variables and three types of binary decision variables. The continuous variables relate to the objective function value or makespan ($C_{\max}$), the processing time (q_{ms}), and completion time (π_{ms}) of items m in workstations s. The binary decision variables relate to the assignments that are to determine whether the task is assigned to a workstation (y_{js}), whether the worker is chosen for a workstation (r_{ws}), and also the three dimensional variables, a_{wjs} which combine the assignment solution of both variables y_{js} and r_{ws}. Different types of constraints are formulated to ensure feasibility of assignment. The first set of constraints represents the mechanism of the flow line in which all items follow the same sequence of operations. The processing time of tasks was used to evaluate the completion time of each item. The second set of constraints involves worker – workstation assignment. They are required to ensure that all workers must be assigned to operate tasks within a workstation. The last set of constraints involves task – workstation assignment. They are required to ensure that only consecutive tasks are grouped and assigned to workstations. The MIP model for this problem was developed using the notations in Table 2:

Index			
M	= set of items	, $m \in M$	,for m = 1, …, i
S	= set of workstations	, $s \in S$	,for s = 1,…, k
J	= set of tasks	, $j \in J$	,for j = 1,…, o
W	= set of workers	, $w \in W$	,for w = 1,…, k
Parameter			
i	= the number of items		
k	= the number of workstations / the number of workers		
o	= the number of tasks		
T_{wmj}	= the processing time of item m task j operated by worker w		
Variable			
a_{wjs}	= 1 if task j is assigned to worker w in workstation s 0 otherwise		
y_{js}	= 1 if task j is assigned in or before workstation s 0 otherwise		
r_{ws}	= 1 if worker *w* works in workstation s 0 otherwise		
q_{ms}	= the processing time of item m in the workstation s		
π_{ms}	= the completion time of item m in workstation s		

Table 2
Notations

Minimize π_{ik} (1)

Subject to

Completion time constraint:

$$q_{ms} = \sum_{w \in W} \sum_{j \in J} T_{wmj} a_{wjs} \qquad \forall m \ , \ \forall s \qquad (2)$$

$$\pi_{11} = q_{11} \qquad (3)$$

$$\pi_{ms} \geq \pi_{ms-1} + q_{ms} \qquad 1 \leq m \leq M \ , \ \ 2 \leq s \leq S \qquad (4)$$

$$\pi_{ms} \geq \pi_{m-1s} + q_{ms} \qquad 2 \leq m \leq M, \ \ 1 \leq s \leq S \qquad (5)$$

Worker – workstation assignment constraint:

$$\sum_{j \in J} \sum_{s \in S} a_{wjs} \geq 1 \qquad \forall w \qquad (6)$$

$$\sum_{j \in J} a_{wjs} \leq o \times r_{ws} \qquad \forall w \ , \ \forall s \qquad (7)$$

$$\sum_{s \in S} r_{ws} = 1 \qquad \forall w \qquad (8)$$

$$\sum_{w \in W} r_{ws} = 1 \qquad \forall s \qquad (9)$$

Grouping task – workstation assignment constraint:

$$y_{ok} = 1 \quad (10)$$

$$y_{j+1s} \le y_{js} \qquad 1 \le j \le J-1 \ , \forall s \quad (11)$$

$$y_{js} \le y_{js+1} \qquad \forall j \ , 1 \le s \le S-1 \quad (12)$$

$$y_{j1} = \sum_{w \in W} a_{wj1} \qquad \forall j \quad (13)$$

$$y_{js} - y_{js-1} = \sum_{w \in W} a_{wjs} \qquad \forall j \ , 2 \le s \le S \quad (14)$$

Objective function (1) is to minimize the makespan. Constraints (2) are used to calculate the sum processing time of all tasks that are assigned in workstation s for item *m*. Constraint (3) reflects that the production line starts empty, no WIP at the beginning of production. Constraints (4) and (5) are used to calculate the completion time of item m in workstation s.

Workstation s can process an item only after the previous workstation $s-1$ has finished the operation on that item and item m can be operated on at a workstation only after the previous item $m-1$ has completed the operation on that workstation. Constraints (6) force all workers to be assigned to at least one task. Constraints (7) represent that if a worker is assigned to any task in a workstation, they must work at that workstation. Constraints (8) and (9) represent one - one assignment between workers and workstations. Constraints (10) through (14) concern grouping tasks and assigning them to workstations. Constraints (10) through (12) force a structure on the task-workstation assignments. Constraint (10) assigns the last task o to the last workstation k. Constraints (11) force the required precedence relationships among the tasks in the *y* variables. That is, if $y_{j+1s} = 1$ then $y_{js} = 1$. (e.g., If $y_{53} = 1$, then $y_{43} = 1$ and, in turn, if $y_{43} = 1$, then $y_{33} = 1$.) Similarly, constraints (12) force the correct structure for the precedence relationships on the workstations: if $y_{js} = 1$ then $y_{js+1} = 1$ (e.g., If $y_{11} = 1$, then $y_{12} = 1$ and since $y_{12} = 1$ then $y_{13} = 1$. Constraints (13) and (14) also ensure that a task cannot be split among workers; thus, each task will be assigned to only one worker.

To illustrate how these variables are interpreted, consider the 5 task, 3 workstations example discussed previously. If tasks 1, 2, and 3 are assigned to workstation 1, task 4 is assigned to workstation 2 and task 5 is assigned to workstation 3, this assignment would be represented in the model as $\vec{y}_1^T = [1\ 1\ 1\ 0\ 0]$, $\vec{y}_2^T = [1\ 1\ 1\ 1\ 0]$ and $\vec{y}_3^T = [1\ 1\ 1\ 1\ 1]$. To interpret these vectors, $y_{11} = y_{21} = y_{31} = 1$ in $\vec{y}_1^T$ means that tasks 1, 2 and 3 are grouped and assigned to workstation 1. For the other 2 workstations (*s*= 2 and 3), the assignments are determined by $y_{js} - y_{js-1}$ for each j. So, for example, $\vec{y}_2^T - \vec{y}_1^T = [0\ 0\ 0\ 1\ 0]$ means that task 4 is assigned to workstation 2.This is how the constraint set restricts consecutive tasks to the same workstation.

III. Solution Approach

The idea behind the heuristic is applying the concept of an upper envelope of a non-buffered system in order to determine the upper bound (UB) and lower bound (LB) of the solution to limit search space. The solutions are searched only between UB and LB. The heuristic starts by grouping tasks into workstations then, assigning workers to perform grouped tasks. Groups of consecutive tasks are generated based on the trial value, which is a predetermined value between UB and LB. The objective of the problem is to determine the assignment of the grouped tasks and workers which has the minimum makespan.

Regarding the relation between idle time and makespan, the idle time has the recurrence relation between the idle time of the previous workstation and the idle time of the previous item on the same workstation. Bellman [15] identified makespan in two simple expressions. For the first expression, makespan is calculated by the idle time on the last station plus the summation of the task's processing time of the last workstation. For the second expression, it is calculated by the flow time of a last item from the completion time on the first workstation to the completion on the last workstation plus the summation of task's processing time of the first station as shown in Figure 1.

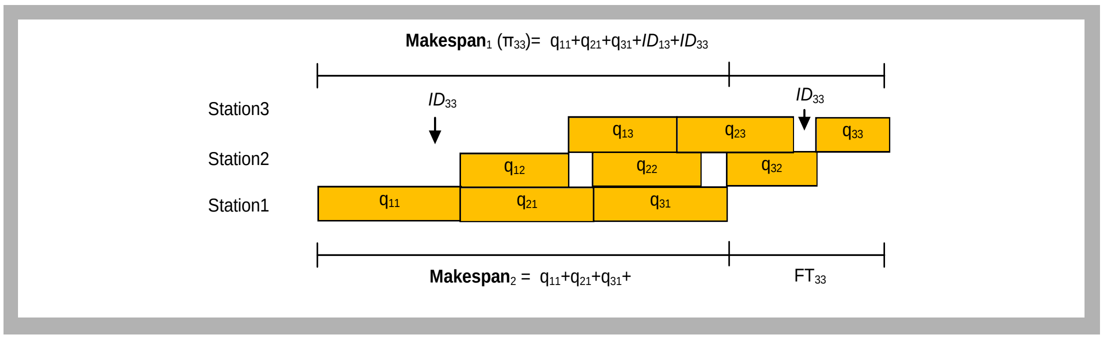


Figure 1
An expression of the Makespan

Makespan depends on the idle time on the last station plus the total task processing time of the last workstation, and the idle time on the last station depends on the idle time of previous stations. The recurrence relation of the idle times of the previous workstation is shown in the appendix. We developed a methodology to minimize the idle time of every workstation via a recurrence relation. The details of the heuristic are presented in the next section

IV. Details of the Heuristic and Numerical Example

The heuristic simplifies the problem by grouping tasks first. After tasks are grouped, the feasible assignment between the groups of tasks and workers will be determined by the modified Dijkstra's algorithm. We developed a procedure to generate groups of tasks. First, the UB and LB of the solution are determined. Then the trial value between UB and LB is set as a bound for task grouping. They are detailed in the following steps. Figure 2 shows the flowchart of the heuristic.

Step 1: Generate the initial UB and LB

To determine UB and LB, the summation of task processing time is determined in Table 3 based on the data in Table 1 and it is solved as an assembly line balancing problem to minimize the maximum workstation time. The objective function value in this stage becomes the LB of the original problem. The solution of worker - task assignment is used to evaluate makespan. This makespan becomes the UB of the original problem.

For example, to minimize the maximum workstation time, the solution of the task-workstation assignment is {12, 3, 45}, the worker-workstation assignment is {C, B, A} and the objective value is 16, so the LB is 16. The makespan of this solution and also the UB is 29, as shown in Figure 3.

Step 2: Set the trial value (TV)

Once LB and UB are determined, a trial value is set. A trial value is chosen to be the bound for generating groups of tasks. The trial value is 10% of the UB-LB difference increased from the LB or TV = LB + (UB-LB)×0.1. Therefore, the first trial value of the example is 16 + (29-16) × 0.1 = 17.3. The UB or the makespan value of the solution consists of the idle time plus workstation time at the last workstation. The LB is the minimum size of grouped tasks. However, the solution of the minimum groups of tasks may not give the optimal solution in makespan value. For this reason, we set the trial value by increasing the LB with the small amount (10%) of the UB-LB difference in order to search for a better solution in a close area.

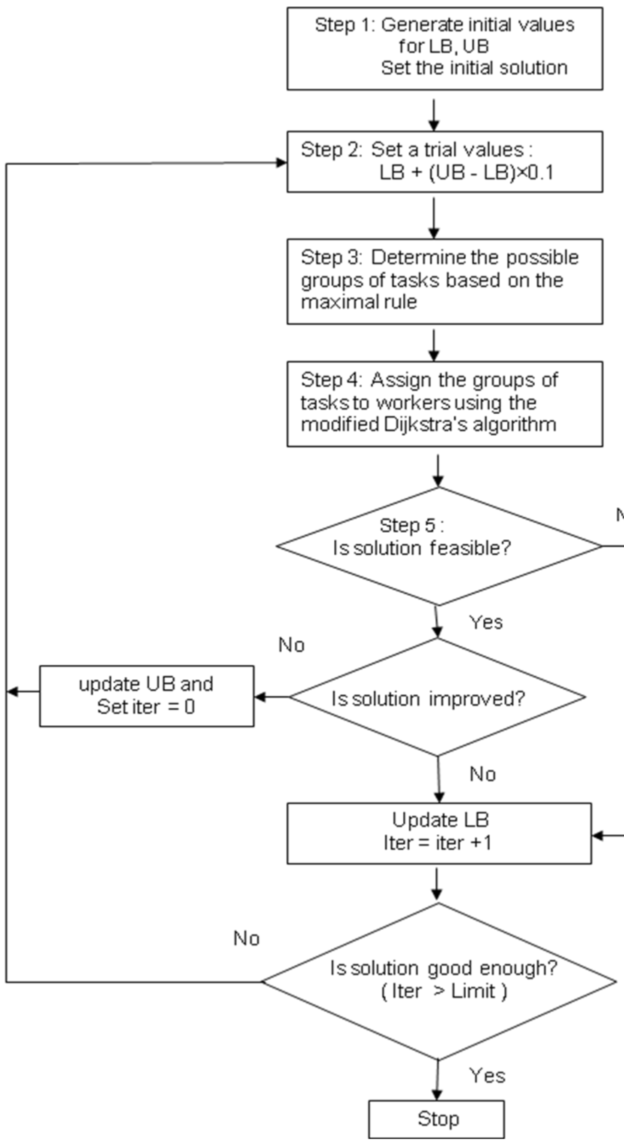

Step 1: Generate initial values for LB, UB Set the initial solution
Step 2: Set a trial values : LB + (UB - LB)×0.1
Step 3: Determine the possible groups of tasks based on the maximal rule
Step 4: Assign the groups of tasks to workers using the modified Dijkstra's algorithm
Step 5 : Is solution feasible?
No
Yes
No
update UB and Set iter = 0
Is solution improved?
No
Update LB Iter = iter +1
No
Is solution good enough? (Iter > Limit)
Yes
Stop

Figure 2
The flowchart of heuristic

Table 3
The summation of task processing time of 3 items of workers (A, B, C)

	task 1	task 2	task 3	task 4	task 5
workerA	11	6	15	4	11
workerB	12	8	16	5	11
workerC	8	8	18	5	10

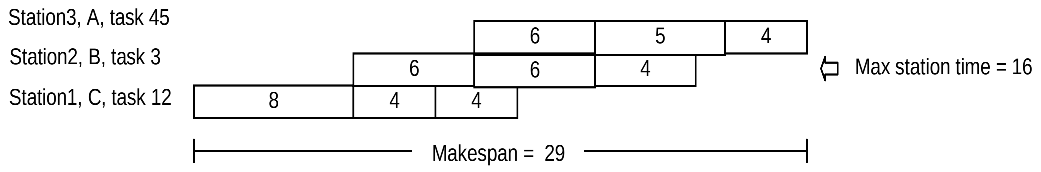


Figure 3
A solution of the example to determine UB and LB

Step 3: Determine the groups of tasks

The idea is that we will generate only the groups of tasks based on the maximal station load rule which is that a workstation will never close if fittable tasks remain. The consecutive tasks will be grouped without exceeding the current trial value and with the maximum group size. Only the groups of tasks based on the maximal station load rule are generated. Beginning with each task, the next consecutive task will be in the groups. Only the maximum group size (the sum of task processing time) within the trial value is considered. The groups of tasks are generated based on the trial value. For example, the first trial value is 17.3. The possible groups of tasks based on the rule of worker *A* are {tasks1, 2}, {task2} {task3} {tasks4, 5}, and {task 5}. Worker *A* does not operate task 1 alone and 4 alone because the sum processing time of tasks 1 and 2 is 17 seconds and the sum processing time of tasks 4 and 5 is 15 seconds which are below the maximum group size of 17.3 seconds. The possible groups of tasks for worker *B* are {task1}, {task2}, {task3}, {tasks4, 5} and {task5}. Worker *B* does not operate tasks 4 alone because the sum processing time of tasks 4 and 5 is 16 seconds which is below the maximum group size of 17.3 seconds. The possible groups of tasks for worker *C* are {tasks1, 2}, {task2}, {tasks 4, 5} and {task5}. Worker *C* cannot operate task 3 since the processing time of task 3 is 18 seconds which is greater than 17.3.

In Figure 4(A), the number in the nodes shows a group of consecutive tasks within the trial value. In this step, a worker who is chosen to carry out each group of tasks is not considered. However, workers who can do those tasks within the trial value are shown at upper right of the nodes. For node 12, workers A and C can be chosen. In the next step, the heuristic will determine the feasible paths. A feasible path must include all tasks using the number of nodes equal to the number of workers. The infeasible nodes are cut off as in Figure 4(B). The feasible path is duplicated based on the workers who can operate tasks in those nodes. Therefore, node 12 will be extracted to node 12A and 12C as shown in Figure 4(C). After that, the step of the modified Dijkstra's algorithm is run to determine the path which leads to the minimum makespan.

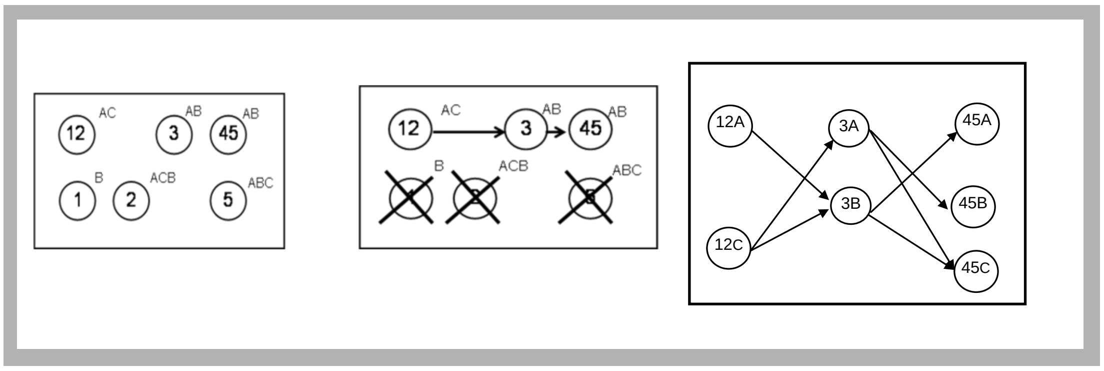


Figure 4(A)
Nodes of the group of tasks

Figure 4(B)
Feasible nodes of task-workstation assignment

Figure 4(C)
Nodes for Modified Dijkstra's algorithm

Step 4: Determine the assignment of the possible groups of tasks and workers

This step applies the idea of Dijkstra's algorithm. Generally, Dijistra's algorithm is an algorithm to determine the shortest path [16]. A node in this problem is a workstation that includes an assignment between the consecutive tasks and a worker and a path means the assignment starting from the first workstation to the last workstation. In the Dijistra's algorithm, the shortest path from the starting node to every other node is determined in order to develop the shortest path to the ending node. In the heuristic, the shortest path to each node becomes the minimum idle time of that node which is designed to search for a solution minimizing makespan. If the minimum idle time of the last station is found, the minimum makespan would tend to be reached.

The search will start at a randomly selected initial node. The initial nodes are the ones which own the first task e.g. node 12A and 12C. The algorithm will repeat at other initial nodes that are not selected. All initial nodes are possible to be chosen.

Figure 6 shows the pseudo code of the modified Dijkstra's algorithm. Firstly, the best makespan is set to UB (line 0). The initial node is randomly selected (line 1) and the variables to keep a solution are initialized (line 2 to line 10). In each step, the heuristic will determine which node in set *Q* which has the minimum idle time. It will be node *u* as shown in line 12. In the first loop, node *u* will be the initial node. If node *u* is found, the node *u* will be added to set *Se* and removed from set *Q* (line 14 to line 15).The nodes which are connected from node *u* using a single arc will be node *b.* The algorithm will check the feasibility of assignment of node *b* from node *u.* The one to one assignment between worker and workstation is examined. If the assignment is feasible (line 17), the sub-makespan and the idle time on the workstation *b* will be determined (line 18).

```
Modified Dijkstra's algorithm
0    Set  best_makespan  = UB
1    Randomly select of initial node
2   FOR   each node v
3             SET   idle time of each node ($idle_v$) = infinite
4             SET   sub makespan of each node ($s\_makespan_v$) = infinite
5             SET   status of the previous node ($previous_v$)= NULL
6   END   Loop
7   SET   $idle_v$ of the selected starting node = 0
8   SET   Se = empty set
9   SET   Q   = set of unselected nodes
10  SET   Have_min = true
11  WHILE   (Have_min  =  true)
12      Determine node (u) which has the minimum idle time in Q,
13      IF node u is found    THEN
14              u   is removed from set Q
15              u   is added in set Se
16          FOR   each node b in Q , a node that is connected from node u  using a  single arc
17            IF (the assignment of b from u valid feasible assignment) THEN
18                Determine idle time, $new\_idle_b$ and sub makespan, $s\_makespan_b$
19                IF (($new\_idle_b$ <  $idle_b$) AND ($s\_makespan_b$ < best_makespan)) THEN
20                    $idle_b$ is replaced by $new\_idle_b$
21                    SET  u  = the $previous_v$ of node b
22                    IF node b is the last workstation
23                        IF $s\_makespan_b$ < best_makespan  THEN  update  best_makespan
24                        END IF
25                    END IF
26                END IF
27            END IF
28          END Loop
29      ELSE Have_min = false   End IF
30  END WHILE
```

Figure 6
The pseudo code of Modified Dijkstra's algorithms

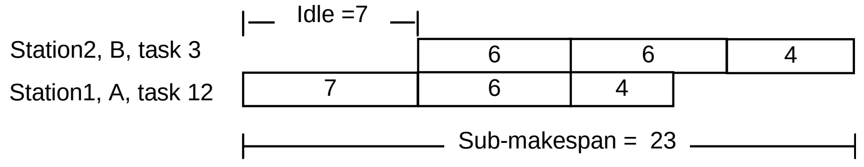


Figure 7
An example of idle and sub-makespan

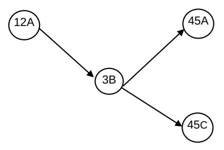
If the new idle time, new_idle_b, is less than idle time, $idle_b$ of node *b* and the sub makespan is less than the best makespan, the idle time of the node, $idle_b$ is updated (line 19 - 20). Then the assignment solution is kept (line 21). For example, the initial node is *12A*, so node *12A* will be node *u*. Node *3B* will be node *b*. This assignment is feasible. The sub-makespan will be 23 and the idle time will be 7. Figure 7 shows an example of the idle and the sub-makespan. Set $previous_b$, the previous node of node *b* equals node *12A*. The steps will repeat for all node *b*. Then the new node *u,* which has minimum idle time, is determined again. The step will repeat until no node having minimum idle is found or idle time of every node in *Q* equals infinity. If the last tasks is in node *b* and $s\_makespan_b$ is less than the best makespan, then the best makespan is updated (line 22-23). For example, node *u* is *3B* and node *45C* is node *b*. Previously, the best makespan was infinity. It is updated to 28 with the assignment solution (*12A*, *3B*, *45C*). Since the heuristic is designed to minimize idle time that is not a direct algorithm to minimize makespan, it is better to keep a set of solutions in a node.

Step 5: Check the improvement of the solution and Stopping Criteria

This step is to select whether UB and LB should be updated in order to further limit search space. If a better makespan of a feasible solution is found, the UB is set to the makespan value in the corresponding solution. Otherwise LB is set to the previous trial value. Then new trial value is re-calculated. The stopping criteria depend on the cumulative number of iterations which is not improving the makespan. For example, the limit of iterations is 2; the heuristic will stop if a solution is not improved in 2 consecutive iterations

doi:10.4186/ej.2010.14.2.1

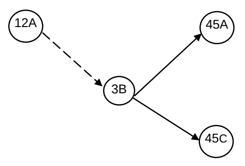
Trial	The Modified Dijkstra's algorithm
The 1st Trial UB=29 LB=16	Trial value = 16 +(29-16) × 0.1 = 17.3 Best makespan =UB = 29



<u>Start at node 12A</u>

Se = {Ø} , *Q* = {12A, 3B, 45A, 45C }
best_makespan = 29

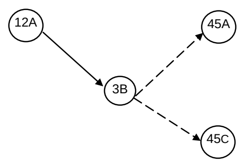
Node	12A	3B	45A	45C
idle$_v$	0	∞	∞	∞
s_makespan$_v$	∞	∞	∞	∞
previous$_v$	-	-	-	-



Node u = *12A*

Se = {12A} , Q = { 3B, 45A, 45C}
best_makespan = 29

		Node b		
Node	12A	3B	45A	45C
idle$_v$	0	7	∞	∞
s_makespan$_v$	-	23	-	-
previous$_v$	-	12A	-	-

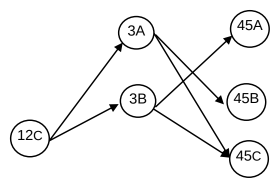


Node u = 3B

Se = {12A, 3B} , Q = { 45A, 45C}

			Node b^1	Node b^2
Node	12A	3B	45A	45C
idle$_v$	0	7	∞	13
s_makespan$_v$	-	23	-	28
previous$_v$	-	12A	-	3B

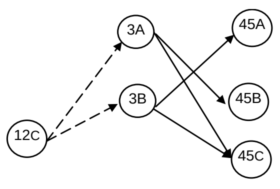
Node b = 45A : Infeasible assignment solution(12A,3B,45A)
Node b = 45C : assignment solution (12A,3B,45C)
best_makespan = 28



<u>Start at node 12C</u>

Se = {Ø} , Q = {12A, 3A, 3B, 45A, 45B, 45C }
best_makespan = 28

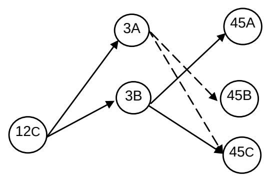
Node	12C	3A	3B	45A	45B	45C
idle$_v$	0	∞	∞	∞	∞	∞
s_makespan$_v$	∞	∞	∞	∞	∞	∞
previous$_v$	-	-	-	-	-	-



Node u = 12C

Se = {12C} , Q = { 3A, 3B, 45A, 45B, 45C}
best_makespan = 28

		Node b^1	Node b^2			
Node	12C	3A	3B	45A	45B	45C
idle$_v$	0	8	8	∞	∞	∞
s_makespan$_v$	-	23	24	-	-	-
previous$_v$	-	12C	12C	-	-	-

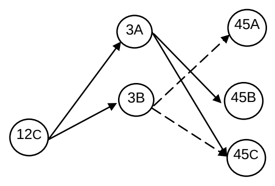


Node u = 3A

Se = {12C,3A}, Q = { 3B, 45A, 45B, 45C}
best_makespan = 28

					Node b^1	Node b^2
Node	12C	3A	3B	45A	45B	45C
idle$_v$	0	8	8	∞	15	∞
s_makespan$_v$	-	23	24	-	31	-
previous$_v$	-	12C	12C	-	3A	-

Node b^2 = 45C : Infeasible assignment (12C,3A,45C)



Node u = 3B

Se = {12C,3A,3B }, Q = { 45A, 45B, 45C}
best_makespan = 28

				Node b^1		Node b^2
Node	12C	3A	3B	45A	45B	45C
idle$_v$	0	8	8	14	15	∞
s_makespan$_v$	-	23	24	29	31	-
previous$_v$	-	12C	12C	3B	3A	-

The best makespan = 28
The best solution is {12A,3B, 45C}

Trial	The Modified Dijkstra's algorithm
The 2nd UB=28 LB=17.3	Trial value = 17.3+ (28-17.3)×0.1=18.37 An improvement of solution is not found.
The 3rd UB=28 B=18.37	Trial value = 19.33 An improvement of solution is not found.
Stop	If the limit number of the consecutive trial value equals 2, the search will stop.

Figure 8
An example of the Modified Dijkstra's algorithm

V. Computational Result

A heuristic algorithm was implemented using the C++ programming language. The quality of the solution and computational time of the proposed heuristic were compared to those obtained from the mathematical model which was solved by CPLEX 8.0. Both CPLEX and the heuristic algorithm were run on a PC with an Intel Core^{TM}2 Duo 2.00 GHz CPU and 1.93 GB of RAM. Normally in a modular production in the garment industry, the number of workers is between 6 and 15. However, only the problem with 7-9 workers was tested since a problem size of 10-15 workers is too large to achieve optimal solutions using the mathematical model. The problem parameters were chosen to reflect a realistic situation in the garment industry. In this testing, the number of tasks was three times the number of workers, since generally in the garment industry a worker is assigned less than three tasks. The numbers of items that were examined were 100 and 300 for the difference of lot sizes.

Table 4 shows all the problem sets of this test. Each problem includes 10 instances. The learning ability of each worker was set based on a Log-Linear model $t_n = t_1 \cdot n^{(\log\phi/\log 2)}$, where t_1 and t_n represent the task processing times of the first and the nth item, and Ø is the learning slope [1]. In the experiment, the learning slope was uniformly generated from the interval between [0.70, 0.90] based on a learning slope for apparel manufacture reported by Dar-El [2]. Since the task processing of the sewing process varies, it was uniformly generated. The mean task processing times of the first item (t_1) were generated uniformly in the interval between [1, 30]. The differences in performance of experienced and inexperienced workers in the sewing line are represented by the percent deviations from the mean task processing times. The percent deviations are uniformly distributed in the interval [-20%, 20%]. For example, if the mean task processing time of the first item is 10 and the percentage of deviation 20%, t_1 becomes 12. Both the mean task processing times and their deviations come from the experience of experts in the field.

Table 4
Problem size

Problem Code	Number of Worker	Number of Task	Number of item	Instances
7w21t100i	7	21	100	10
7w21t300i	7	21	300	10
8w24t100i	8	24	100	10
8w24t300i	8	24	300	10
9w27t100i	9	27	100	10
9w27t300i	9	27	300	10

For the problem sets 7w21t100i, 7w21t300i, 8w24t100i and 8w24t300i, CPLEX was designed to run until the optimal solution was found. Table 5 shows the computational time in the second column. Since the problem size of 9 workers is quite large to achieve the optimal solutions, the run time of the problem was limited. For 9 workers with 100 items, 9w27t100i, the limited run time was 259,200 sec. or (3days). For 9 workers with 300 items, 9w27t300i, the limited run time was 345,500 sec. or (4days) due to the bigger problem. The third column in Table 5 shows the computational time of the heuristic. It was found that the computational time for the heuristic was short compared to the computational time from CPLEX. The optimal value or makespan from CPLEX was compared the final solution from the heuristic. Nine tests of 9w27t100i problem and 4 tests of 9w27t300i found the exact solution within the limit time of each problem. The fourth and fifth column of Table 5 shows the percentage difference in the makespan of the heuristic solution versus the optimal solution, when available. For problems 7, 8 and 9 workers, the heuristic found a solution within 3% of the optimal solutions with 0.67% on average. The heuristic clearly outperforms CPLEX. The experimental results show that the heuristic provides good solutions with a better CPU time compared to the CPLEX solutions. When the number of items increases, the CPU time of heuristic is slightly increased.

doi:10.4186/ej.2010.14.2.1

Test	CPU. Optimal (sec.)	CPU. Heu. (sec.)	% diff makespan	
			Avg.	Max.
7w21t100i	951.15	4.44	0.07	0.56
7w21t300i	8,669.15	9.40	0.35	2.93
8w24t100i	11,847.81	23.15	0.36	2.15
8w24t300i	106,111.15	32.30	0.09	0.58
9w27t100i	-*	83.94	0.54(9)	1.49(9)
9w27t300i	-**	188.93	0.685(4)	0.825(4)

* Limited run time = 2 days, ** Limited run time = 3 days

Table 5
The CPU. Time on average of optimal sol. and heuristic sol.

VI. Conclusions

This paper has proposed a heuristic methodology for the task-worker assignment problem with varying learning slopes for the fashion industry. The problem was formulated in an integer linear programming model with the objective of minimizing makespan. For solving the problem, a heuristic was developed. The LB and UB were determined. Tasks were grouped based on the trial value between LB and UB following the maximal load rule. The worker-workstation assignment solution was determined based on the relation between idle time and makespan. A modified Dijkstra's algorithm was developed to determine the most efficient worker-workstation assignment.

The effectiveness of the heuristic was evaluated in terms of computational time and quality of the solution. The experimental results show that the heuristic provides good solutions with a better CPU time compared to the optimal solutions. For problems with 7, 8 and 9 workers, the heuristic found a solution within 3% from the optimal solutions with 0.67% on average.

VI. Appendix

The appendix shows the recurrence relation between the idle times of the previous workstation. Let FT_{ms} be the flow time of an item m from the completion time of the first workstation to the completion time of workstation s as shown in equation 15 and let ID_{ms} be the idle time of an item m at workstation s, there are two types of expression of the makespan. Using the idle time of the last workstation, the makespan (π_{ik}) = $\sum_{m=1}^{i} q_{mk} + \sum_{m=1}^{i} ID_{mk}$,the first term of the right side is the sum of the task processing time and the second term is the sum of the idle times at the last workstation. Using the flow time, the makespan (π_{ik}) = $\sum_{m=1}^{i} q_{m,1} + FT_{ik}$, the first term of the right side is the sum of the task processing time at the first workstation and flow time of the last item at the last workstation.

$$FT_{ms} = \pi_{ms} - \pi_{m1} \tag{15}$$

Since π_{ms} = $\sum_{z=1}^{m} q_{zs} + \sum_{z=1}^{m} ID_{zs}$ and π_{m1} = $\sum_{z=1}^{m} q_{z1}$

, so $$FT_{ms} = \sum_{z=1}^{m} ID_{zs} + \sum_{z=1}^{m} (q_{zs} - q_{z1}) \tag{16}$$

Since the start of the processing of item m at workstation s follows both the completion of item m at the previous workstation $s-1$ and the completion of the previous item $m-1$ on workstation s, the recurrence relation holds for every flow time, so

$$FT_{ms} = q_{ms} + \max[FT_{m-1s} - q_{m1}, FT_{ms-1}] \tag{17}$$

By replacing FT_{m-1s} and FT_{ms-1} in an eq. 17 by the eq. 16,

$$\sum_{z=1}^{m} ID_{zs} + \sum_{z=1}^{m} (q_{zs} - q_{z1}) =$$
$$q_{ms} + \max[\sum_{z=1}^{m-1} ID_{zs} + \sum_{z=1}^{m-1} (q_{zs} - q_{z1}) - q_{m1}, \sum_{z=1}^{m} ID_{zs-1} + \sum_{z=1}^{m} (q_{zs-1} - q_{z1})] \tag{18}$$

By subtracting the quantity, $\sum_{z=1}^{m} (q_{zk} - q_{z1})$ from both sides of the eq. 18, the recurrence relation for idle time is obtained.

$$\sum_{z=1}^{m} ID_{zs} = \max[\sum_{z=1}^{m-1} ID_{zs}, \sum_{z=1}^{m} ID_{zs-1} + (\sum_{z=1}^{m} q_{zs-1} - \sum_{z=1}^{m-1} q_{zs})] \tag{19}$$

An eq.19 shows the recurrence relation between the idle times of the previous workstation.

doi:10.4186/ej.2010.14.2.1

REFERENCES

[1] T. P. Wright, "Factors affecting the cost of airplanes," *Journal of the Aeronautical Sciences,* vol. 3, no. 4, pp. 122-128, 1936.

[2] E. Dar-El, *Human Learning: From Learning Curves to Learning Organizations*. Springer, 2000.

[3] G. Mosheiov, "Scheduling problems with a learning effect," *European Journal of Operational Research,* vol. 132, no. 3, pp. 687-693, 2001.

[4] A. K. Chakravarty, "Line balancing with task learning effects," *IIE Transactions (Institute of Industrial Engineers),* vol. 20, no. 2, pp. 186-193, 1988.

[5] D. A. Nembhard and N. Osothsilp, "Learning and forgetting-based worker selection for tasks of varying complexity," *Journal of the Operational Research Society,* vol. 56, no. 5, pp. 576-587, 2005.

[6] R. Karni and Y. T. Herer, "Allocation of tasks to stations in small-batch assembly with learning: basic concepts," *International Journal of Production Research,* vol. 33, no. 11, pp. 2973 - 2998, 1995.

[7] M. D. Toksari, S. K. Isleyen, E. Guner, and O. F. Baykoc, "Simple and U-type assembly line balancing problems with a learning effect," *Applied Mathematical Modelling,* vol. 32, no. 12, pp. 2954-2961, 2008.

[8] Y. Cohen and M. E. Dar-El, "Optimizing the number of stations in assembly lines under learning for limited production," *Production Planning & Control,* vol. 9, no. 3, pp. 230 - 240, 1998.

9] Y. Cohen, G. Vitner, and S. C. Sarin, "Optimal allocation of work in assembly lines for lots with homogenous learning," *European Journal of Operational Research,* vol. 168, no. 3, pp. 922-931, 2006.

[10] Y. Cohen, G. Vitner, and S. C. Sarin, "Work allocation to stations with varying learning slopes and without buffers," *European Journal of Operational Research,* vol. 184, no. 2, pp. 797-801, 2008.

[11] Y. Cohen and M. E. Dar-El, "Optimizing the number of stations in assembly lines under learning for limited production," *Production Planning & Control,* vol. 9, no. 3, pp. 230 - 240, 1998.

[12] Y. Cohen and M. E. Dar-El, "Optimizing the number of stations in assembly lines under learning for limited production," *Production Planning and Control,* vol. 9, no. 3, pp. 230-240, 1998.

[13] R. G. Askin and C. R. Standridge, *Modeling and Analysis of Manufacturing Systems*. New York: John Wiley & Sons, Inc., 1993.

[14] F. Altimparmak, B. Dengiz, and A. A. Bulgak, "Buffer allocation and performance modeling in asychronous assembly system operations: An artificial neural network metamodeling approach," *Applied Soft Computing,* vol. 7, no. 3, pp. 946-956, 2007.

[15] R. Bellman, A. O. Esogbue, and I. Nabeshima, *Mathematical Aspects of Scheduling & Applications.* New York: Pergamon Press, 1982.

[16] S. S. Skiena, *The Algorithm Design Manual.* New York: Springer Verlag, 1998.