Two machine flow shop scheduling problem with no wait in process: Controllable machine speeds

Vitaly A. Strusevich

Erasmus University, P.O. Box 1738, 3000 DR, Rotterdam, The Netherlands

Received 5 July 1989; revised 31 August 1993

Abstract

The paper considers a two machine flow shop scheduling problem with no wait in process. The objective is to find the optimal values of machine speeds and to determine the time-optimal schedule. There are two machines, A and B, and a set of n jobs each consisting of two operations. The first operation of each job is to be processed on machine A and the second on machine B. For each feasible schedule, the second operation of each job starts at the same time the first operation of this job is completed (the no wait condition). Machine speeds are supposed to be controllable and should be selected in order to minimize a certain penalty function. The objective function depends on both profits resulting from the reduction of the makespan and expenditures for increasing the machine speeds. An $O(n^3)$ algorithm is described to solve the problem.

1. Introduction and the problem formulation

The paper considers a generalized version of the two machine flow shop scheduling problem with no wait in process which can be formulated as follows.

There are two machines, A and B, and a set $N = \{1, 2, \ldots, n\}$ of jobs each consisting of two operations. The first operation of each job is to be processed on machine A and the second on machine B. For each feasible schedule, the second operation of each job starts at the same time the first operation of this job is completed, i.e., the no wait condition should not be violated.

Traditionally, in two machine flow shop scheduling models, it is assumed that processing each job $i$ on machines A and B takes $a_i > 0$ and $b_i > 0$ time, respectively, and those processing times are given. In the model under consideration the values $a_i$ and $b_i$ are still given. However, actual processing times depend on the machine speeds which are not known in advance but must be chosen, so that the original values $a_i$ and $b_i$ can be either reduced or enlarged. Formally, if the speed of machine A is $v_A > 0$, then processing job $i \in N$ on that machine takes $w_i = 1/v_A$ times of the
original value \(a_i\), i.e., \(w_Aa_i\) time units. Similarly, if the speed of machine \(B\) is \(v_B > 0\), then processing job \(i \in N\) on that machine takes \(w_Bb_i\) time units where \(w_B = 1/v_B\).

Let \(v_A\) and \(v_B\) be the fixed values of speeds of machines \(A\) and \(B\), respectively. As discussed in [5] and [13], for each no wait schedule preemption is not allowed and, moreover, both machines process the jobs of set \(N\) according to the same sequence \(\pi = (i_1, i_2, \ldots, i_n)\). Let \(t(v_A, v_B, \pi)\) denote the makespan, i.e., the maximum completion time, for a certain schedule \(s = s(v_A, v_B, \pi)\). If \(\pi^*\) is such a sequence of jobs that \(t(v_A, v_B, \pi^*) \leq t(v_A, v_B, \pi)\) holds for every sequence \(\pi\), then we denote \(t(v_A, v_B) = t(v_A, v_B, \pi^*)\).

The problem we consider here is to find values \(v_A^*\) and \(v_B^*\) of machine speeds \(v_A\) and \(v_B\), respectively, which minimize the function

\[
F(v_A, v_B) = c_0 t(v_A, v_B)^{q_1} + c_1 v_A^{q_2} + c_2 v_B^{q_2}. \tag{1.1}
\]

Here \(c_0, c_1\) and \(c_2\) are given positive constants, while \(q_1\) and \(q_2\) are given positive rationals. The function \(F(v_A, v_B)\) depends on both profits resulting from the reduction of the makespan and expenditures for increasing the machine speeds. After the values \(v_A^*\) and \(v_B^*\) have been found it is necessary to find the corresponding optimal schedule.

Problems similar to that under consideration were studied in [7, 16] (for the two machine open shop), in [6] (for the two machine mixed shop), and in [14, 15] (for the two machine flow shop). Note that the results from [14] and [16] cannot be improved from the point of view of the running time of the developed algorithms since in both cases solving a problem with controllable machine speeds takes no more time than finding a time-optimal schedule in the case of constant speeds.

Problems similar to that under consideration were studied in [7, 16] (for the two machine open shop), in [6] (for the two machine mixed shop), and in [14, 15] (for the two machine flow shop). Note that the results from [14] and [16] cannot be improved from the point of view of the running time of the developed algorithms since in both cases solving a problem with controllable machine speeds takes no more time than finding a time-optimal schedule in the case of constant speeds.

Problems similar to that under consideration were studied in [7, 16] (for the two machine open shop), in [6] (for the two machine mixed shop), and in [14, 15] (for the two machine flow shop). Note that the results from [14] and [16] cannot be improved from the point of view of the running time of the developed algorithms since in both cases solving a problem with controllable machine speeds takes no more time than finding a time-optimal schedule in the case of constant speeds.

Note that the ordinary two machine flow shop scheduling problem with no wait in process to minimize the makespan, i.e., the problem with \(v_A = v_B = 1\), is solvable in \(O(n \log n)\) time [3, 4, 9, 10, 17] because it can be reduced to the Gilmore–Gomory traveling salesman problem [1-4]. On the other hand, if some of \(a_i\) and/or \(b_i\) are allowed to be zero and zero processing times imply that the corresponding operations should not be performed, then the ordinary problem is NP-hard in the strong sense [12]. If there are not two but \(M > 2\) machines and the processing times are strictly positive, then the ordinary problem was shown to be NP-hard in the strong sense, for \(M\) variable in [8], for fixed \(M \geq 4\) in [9] and, finally, for \(M \geq 3\) [11].

The main result of this paper is an \(O(n^3)\) algorithm for finding the values \(v_A^*\) and \(v_B^*\) of machine speeds which minimize the objective function (1.1).

The paper consists of five sections. Section 2 introduces an important function \(t_0(y)\) and presents an algorithm for finding the optimal values of the machine speeds under the assumption that the breakpoints of that function are known. Section 3 examines the situation of the fixed values of machine speeds and establishes its connection with the Gilmore–Gomory traveling salesman problem. Section 4 describes a method for finding the breakpoints of function \(t_0(y)\). The general algorithm is presented in Section 5.
2. The Ishii–Nishida method. Finding the optimal speeds

At first, we assume that \( v_A \) and \( v_B \) are the fixed values of speeds of machines \( A \) and \( B \), respectively.

Let \( \pi = (i_1, i_2, \ldots, i_n) \) be an arbitrary sequence of the jobs of set \( N \). This sequence specifies a schedule \( s = s(v_A, v_B, \pi) \) with the makespan \( t(v_A, v_B, \pi) \). It follows from [10, 17] that

\[
t(v_A, v_B, \pi) = \sum_{k=1}^{n-1} \max\{w_B b_{i_k} - w_A a_{i_{k+1}}, 0\} + w_A \sum_{k=1}^{n} a_{i_k} + w_B b_{i_n}. \tag{2.1}
\]

In order to find a sequence \( \pi^* \) such that \( t(v_A, v_B, \pi^*) \leq t(v_A, v_B, \pi) \) holds for each sequence \( \pi \) of jobs, it is sufficient to minimize the function

\[
t_0(v_A, v_B, \pi) = \sum_{k=1}^{n-1} \max\{w_B b_{i_k} - w_A a_{i_{k+1}}, 0\} + w_B b_{i_n}
\]

over the set of all sequences of jobs.

Given the fixed values \( v_A \) and \( v_B \), we define \( \gamma = v_B/v_A = w_A/w_B \). Then we get

\[
t_0(v_A, v_B, \pi) = \sum_{k=1}^{n-1} \max\{b_{i_k} - \gamma a_{i_{k+1}}, 0\} + b_{i_n}. \tag{2.2}
\]

Given fixed \( \pi = (i_1, i_2, \ldots, i_n) \) and \( \gamma \), one can determine subsets of jobs

\[
N_{a}(\gamma, \pi) = \{i_{k+1} | k = 1, 2, \ldots, n-1, b_{i_k} > \gamma a_{i_{k+1}}\}
\]

and

\[
N_{b}(\gamma, \pi) = \{i_k | k = 1, 2, \ldots, n-1, b_{i_k} > \gamma a_{i_{k+1}}\}.
\]

Then relation (2.2) becomes

\[
t_0(\gamma, \pi) = \gamma x_0(\gamma, \pi) + \beta_0(\gamma, \pi),
\]

where

\[
x_0(\gamma, \pi) = - \sum_{i_{k+1} \in N_{a}(\gamma, \pi)} a_{i_{k+1}},
\]

\[
\beta_0(\gamma, \pi) = \sum_{i_k \in N_{b}(\gamma, \pi)} b_{i_k} + b_{i_n}.
\]

Suppose that \( \gamma \) is variable and we have found a sequence \( \gamma(0), \gamma(1), \ldots, \gamma(r) \) such that

\[
0 = \gamma(0) < \gamma(1) < \cdots < \gamma(r) = W
\]

where \( W \) is a sufficiently large number, and for each

\[
\gamma \in (\gamma(j-1), \gamma(j)], j = 1, 2, \ldots, r,
\]

the same sequence \( \pi^*(j) \) minimizes function \( t_0(\gamma, \pi) \), and, moreover, the sets \( N_{a}(\gamma, \pi^*(j)) \) and \( N_{b}(\gamma, \pi^*(j)) \) are the same. We define a function \( t_0(\gamma) = t_0(\gamma, \pi^*(j)) \). The points \( \gamma(0), \gamma(1), \ldots, \gamma(r) \) will be called the breakpoints of function \( t_0(\gamma) \).
In this section we reduce the original problem to that of finding the breakpoints of function \( \tau_0(y) \).

For each \( y \in (\gamma(j-1), \gamma(j)] \), \( j = 1, 2, \ldots, r \), the following relation

\[
\tau(v_A, v_B) = w_B \tau_0(y) + \gamma \sum_{k=1}^{n} a_k = w_B(\gamma \alpha(j) + \beta(j))
\]

holds. Here \( \beta(j) = \beta_0(\gamma, \pi^*(j)) \) and

\[
\alpha(j) = \alpha_0(\gamma, \pi^*(j)) + \sum_{k=1}^{n} a_k.
\]

Let us rewrite relation (1.1) in the form

\[
F(v_A, v_B) = c_0 \tau(v_A, v_B)^{q_1} + w_B^{-q_2}(c_1 \gamma^{-q_2} + c_2)
\]

and for each \( j, 1 \leq j \leq r \), substitute (2.3) into (2.4). We get the function which is denoted by \( G_j(y, w_B) \). It is of the form

\[
G_j(y, w_B) = c_0 w_B^{q_1}(\gamma \alpha(j) + \beta(j))^{q_1} + w_B^{-q_2}(c_1 \gamma^{-q_2} + c_2).
\]

Thus, the problem of minimizing function (1.1) is reduced to \( r \) subproblems \( P_j \), \( j = 1, 2, \ldots, r \). A subproblem \( P_j \) is that of minimizing function \( G_j(y, w_B) \) for \( y \in [\gamma(j-1), \gamma(j)] \) and \( w_B > 0 \).

To minimize function \( G_j(y, w_B) \), \( 1 \leq j \leq r \), for \( y \in [\gamma(j-1), \gamma(j)] \) and \( w_B > 0 \) we shall use the results of Ishii and Nishida [7] (see also [6]) where practically the same problem was examined.

Let \( \gamma^*(j) \) and \( w_B^*(j) \) be the values of variables \( \gamma \) and \( w_B \), respectively, which minimize function \( G_j(y, w_B) \), \( 1 \leq j \leq r \), for \( y \in [\gamma(j-1), \gamma(j)] \). To find these values, i.e., to solve subproblem \( P_j \), Ishii and Nishida offered the following.

**Algorithm 2.1 (the Ishii–Nishida algorithm).**

1. Calculate \( Q = \{ [c_1 \beta(j)]/[c_2 \alpha(j)] \}^{1/(q_2 + 1)} \).
2. Find \( \gamma^*(j) \), where
   (a) \( \gamma^*(j) = \gamma(j-1) \) if \( \gamma(j-1) \geq Q \); 
   (b) \( \gamma^*(j) = \gamma(j) \) if \( \gamma(j) \leq Q \); 
   (c) \( \gamma^*(j) = Q \) if \( \gamma(j-1) < Q < \gamma(j) \).
3. Calculate

\[
w_B^*(j) = \{ q_2/(c_0 q_1) \} (\gamma^*(j) \alpha(j) + \beta(j))^{-q_1}(c_1 \gamma^*(j)^{-q_2} + c_2)^{1/(q_1 + q_2)}
\]

and Stop.

Note that, if one assumes that the power and root operations can be implemented in constant time, then the values \( \gamma^*(j) \) and \( w_B^*(j) \) can also be found in constant time for each \( j, 1 \leq j \leq r \).
In order to find the desired values \( v_A^* \) and \( v_B^* \) of the machine speeds which minimize function (1.1) we may use the following.

**Algorithm 2.2.**

1. Let all the values \( y^*(j) \) and \( w_B(j) \) be found, \( j = 1, 2, \ldots, r \). Find the values \( y^* \) and \( w_B^* \) such that \( y^* = y^*(k) \) and \( w_B^* = w_B(k) \) for a certain \( k, 1 \leq k \leq r \), where

\[
G_k(y^*(k), w_B^*(k)) = \min \{G_j(y^*(j), w_B(j)) | j = 1, 2, \ldots, r\}.
\]

2. Set \( v_B^* = 1/w_B^* \) and \( v_A^* = v_B^*/y^* \). Stop.

If one assumes that all the values \( G_j(y^*(j), w_B^*(j)) \), \( j = 1, 2, \ldots, r \), are known in advance, then the running time of Algorithm 2.2 is \( O(r) \).

Thus, in order to solve the original problem we need to find the breakpoints of the function \( t_0(y) \) and to determine the values \( \alpha(j) \) and \( \beta(j) \) for all \( j, j = 1, 2, \ldots, r \).

3. **Fixed values of the machine speeds**

3.1. **The Gilmore–Gomory algorithm**

In this section we assume that the machine speeds have fixed values; more precisely, \( y \) is supposed to be fixed.

If \( y \) is fixed then the problem of minimizing \( t_0(y, \pi) \) can be reduced to the traveling salesman problem (TSP) similar to the way it was done in \([10, 17]\) for the ordinary case \( (y = 1) \).

Recall that the TSP is a problem of determining a tour of the minimal length visiting each of a certain set of cities exactly once, a matrix of the distances between the cities being given.

We introduce the TSP with \( n + 1 \) cities numbered by integers from 0 to \( n \) and the distance matrix \( D = (D_{pq}) \), where

\[
D_{pp} = + \infty, \quad D_{0q} = 0, \quad D_{pq} = b_p, \quad D_{pq} = \max \{b_p - y a_q, 0\},
\]

\[
p = 0, 1, \ldots, n; \quad q = 0, 1, \ldots, n; \quad p \neq q. \quad (3.1)
\]

Let \( \pi^* = (0, j_1, j_2, \ldots, j_n) \) be the sequence of cities which specifies the optimal tour for this TSP. As follows from \([10, 17]\), the sequence \( \pi^* = (j_1, j_2, \ldots, j_n) \) is the desired sequence which minimizes function \( t_0(y, \pi) \) and, therefore, function \( t(v_A, v_B, \pi) \) for fixed \( v_A \) and \( v_B \).

It is easy to show that the TSP with the distance matrix \( D \) of the form (3.1) is a special case of the well-known Gilmore–Gomory problem [1–4]. The latter problem
is a TSP with the distance matrix of the form

\[ D_{pp} = +\infty, \quad p = 0, 1, \ldots, n, \]

\[ D_{pq} = \int_{A_p}^{A_q} u_1(x) \, dx \quad \text{if } A_q \geq B_p, \]

\[ D_{pq} = \int_{A_q}^{B_p} u_2(x) \, dx \quad \text{otherwise,} \]

\[ p = 0, 1, \ldots, n; \quad q = 0, 1, \ldots, n; \quad p \neq q. \quad (3.2) \]

Here \( u_1(x) \) and \( u_2(x) \) are integrable functions such that \( u_1(x) + u_2(x) \geq 0 \), and \( A_p, B_p, p = 0, 1, \ldots, n \), are given numbers.

If we set \( a_0 = b_0 = 0, A_p = \gamma a_p, B_p = b_p, p = 0, 1, \ldots, n, \) and \( u_1(x) \equiv 0, u_2(x) \equiv 1, \) then formulae (3.2) and (3.1) will coincide.

The Gilmore–Gomory problem was first introduced in [3], and the running time of the solution algorithm was \( O(n^2) \) [2, 3]. Later, it was noted in [9] that the algorithm from [3] could be modified so as to achieve \( O(n \log n) \) running time. The most advanced version of the Gilmore–Gomory algorithm, also of \( O(n \log n) \) running time, is described in [4]. Note that a polynomial-time algorithm for recognizing whether the Gilmore–Gomory conditions (3.2) are fulfilled for an arbitrary matrix is developed in [1].

In what follows, we deal with some specific features of the Gilmore–Gomory algorithm, so we now describe its main steps, assuming that \( u_1(x) = 0 \) and \( u_2(x) \leq 1 \) since only that special case is analyzed below. When describing the algorithm, we follow [4]. However, in order to avoid introducing additional definitions and information on multiplication of permutations and the theory of subtour patching, we use another notation and terminology. The reader interested in the justification of the algorithm may find all the relevant details in [4].

Suppose we are given numbers \( A_p \) and \( B_p, p = 0, 1, \ldots, n, \) and that \( u_1(x) \equiv 0, u_2(x) \equiv 1. \) Without loss of generality, we may assume that \( B_0 \leq B_1 \leq \cdots \leq B_n; \) otherwise, the cities can be renumbered in the required way in \( O(n \log n) \) time.

Algorithm 3.1 (the Gilmore–Gomory algorithm).
1. Find a sequence \( \phi = (m_0, m_1, \ldots, m_n) \) such that \( A_{m_0} \leq A_{m_1} \leq \cdots \leq A_{m_n}. \)
2. Calculate

\[ \delta_q = \max\{\min\{B_{q+1}, A_{m_{q+1}}\} - \max\{B_q, A_{m_q}\}, 0\}, \quad q = 0, 1, \ldots, n-1. \quad (3.3) \]
3. Construct a nondirected graph \( G_\phi \) with the set of vertices \( \{0, 1, \ldots, n\} \), vertices \( p \) and \( q, p \neq q, \) being adjacent if and only if \( p = m_q. \)
4. If the graph \( G_\phi \) is connected, then denote \( \psi = (m_0, m_1, \ldots, m_n), \) and go to Step 8. Otherwise, construct a (multi)graph \( G^*_\phi \) by reducing each connected component of the graph \( G_\phi \) into a single vertex followed by connecting its vertices in the
following way. An edge \( e_q \) of the weight \( \delta_q \) connects two vertices of the graph \( G^1_\phi \) if they correspond to two connected components of the graph \( G_\phi \) such that the vertex \( q \) belongs to one of these components while the vertex \( q + 1 \) belongs to the other.

5. In the graph \( G^1_\phi \), find a spanning tree of minimal total weight. Let \( T \) denote the list of edges which form the tree.

6. Form a sequence \( E \) of the edges of \( T \) in the following way. Start with \( E \) being empty. For each \( q \) from 0 to \( n - 1 \) check whether \( e_q \) belongs to \( T \). If it does not, then take the next edge. Otherwise, if \( A_{m_q} < B_q \) then put \( e_q \) before the first current element of \( E \); if \( A_{m_q} \geq B_q \) then put \( e_q \) after the last current element of \( E \); then proceed with the next edge. Let \( E = (e_{u_1}, e_{u_2}, \ldots, e_{u_l}) \) where \( l \) is the number of edges in \( T \).

7. Start with the sequence \((0, 1, \ldots, n)\). Interchange \( u_1 \) and \( u_1 + 1 \). In the resulting sequence interchange \( u_1 \) and \( u_2 + 1 \) and so on until \( u_1 \) and \( u_i + 1 \) are interchanged. In the resulting sequence interchange \( i \) and \( m_i \) according to the sequence \( \phi \). Let \( \psi = (q_1, q_2, \ldots, q_{n+1}) \) be the obtained sequence.

8. Find the optimal tour which is specified by the sequence \( \tau^* = (0, j_1, j_2, \ldots, j_n) \) where \( j_1 = q_1, j_k = q_{j_{k-1}} \), \( k = 2, 3, \ldots, n \). Stop.

It is easy to verify that Step 1 of the algorithm requires \( O(n \log n) \) time while each of the remaining steps can be done in \( O(n) \) time.

4. Finding the breakpoints of function \( t_0(\gamma) \)

The function \( t_0(\gamma) \) has breakpoints of two types. First, while \( \gamma \) is varying, a sequence which minimizes this function may change; let us call the values of \( \gamma \) where it happens the breakpoints of the first type. Secondly, if for \( \gamma \in (\gamma^1, \gamma^2] \) the same sequence \((i_1, i_2, \ldots, i_n)\) is optimal, then, while \( \gamma \) is varying within that interval the signs of expressions \( b_{i_k} - \gamma a_{i_k+1} \) may change for some \( k, k = 1, 2, \ldots, n - 1 \); the corresponding breakpoints will be called the breakpoints of the second type.

Due to the nature of the function \( t_0(\gamma) \), it is convenient to deal with the related problem of changes of the optimal tour for the corresponding Gilmore–Gomory TSP instead of examining possible changes of sequences.

To do this, we must study the behaviour of the Gilmore–Gomory algorithm in the case when \( a_0 = b_0 = 0, A_p = \gamma a_p, B_p = b_p, p = 0, 1, \ldots, n, \) and \( u_1(x) \equiv 0, u_2(x) \equiv 1, \gamma \in [0, + \infty) \) being variable.

We assume that \( b_0 \leq b_1 \leq \cdots \leq b_n \), so that for the sequence \( \phi = (m_0, m_1, \ldots, m_n) \) one has \( a_{m_0} \leq a_{m_1} \leq \cdots \leq a_{m_n} \).

Note that in the case under consideration \((3.3) \) becomes

\[
\delta_q = \max \{ \min \{ b_{q+1}, \gamma a_{m_q+1} \} - \max \{ b_q, \gamma a_{m_q} \}, 0 \}.
\]
Thus, when $\gamma$ is fixed a city $q$, $q = 0, 1, \ldots, n - 1$, has to belong to one of the following six types:

1. $\gamma a_m \leq \gamma a_{m+1} \leq b_q \leq b_{q+1}, \delta_q = 0$,
2. $\gamma a_m \leq b_q < \gamma a_{m+1} \leq b_{q+1}, \delta_q = \gamma a_{m+1} - b_q$,
3. $b_q < \gamma a_m \leq b_{q+1} < \gamma a_{m+1}, \delta_q = b_{q+1} - \gamma a_m$,
4. $b_q < \gamma a_m \leq b_{q+1} < b_{q+2}$,
5. $b_q < b_{q+1} < \gamma a_m$,
6. $b_q < b_{q+1} < b_{q+2}$.

Let us consider Algorithm 3.1 and find out which steps of that algorithm can produce different results for different values of $\gamma$. Definitely, one of those steps is Step 2 since $\gamma$ is involved in calculating the values $\delta_q$. Then, to find a minimum spanning tree in Step 5 one has to compare the values $\delta_q$, and those depend on $\gamma$. Finally, in Step 6 the edges of the list $T$ are treated differently with respect to the sign of the difference $\gamma a_m - b_q$.

Suppose we know the tour which is optimal for a certain $\gamma$. One may conclude that this tour remains optimal while $\gamma$ is varying if the following three conditions are satisfied:

(i) the type of each city remains the same (Step 1);
(ii) the edges belonging to the list $T$ remain the same (Step 5);
(iii) the sign of expression $\gamma a_m - b_q$ for each edge $e_q$ belonging to $T$ is not changed (Step 6).

For our purposes, it is desirable to formulate condition (ii) in another way.

The sequence of the cities sorted in nondecreasing order with respect to $\delta_q$, $q = 0, 1, \ldots, n - 1$, will be called the $\delta$-sequence. Suppose that the greedy algorithm is applied to find the minimum spanning tree in the graph $G_\phi$. Recall that the greedy algorithm considers the edges $e_q$ according to the $\delta$-sequence, and the next edge is included into the current list $T$ unless it produces no cycles with the edges in $T$. Thus, if the $\delta$-sequence of the cities does not change while $\gamma$ is varying, then the list $T$ remains the same. Thus, we have the following lemma.

**Lemma 4.1.** If the $\delta$-sequence of the cities $0, 1, \ldots, n - 1$ does not change while $\gamma$ is varying within an interval $[\gamma^1, \gamma^2]$, then condition (ii) is satisfied.

Let $p$, $0 \leq p \leq n - 1$, be a city of type $x$, $1 \leq x \leq 6$, and $q$, $0 \leq q \leq n - 1$, $p \neq q$, be a city of type $y$, $1 \leq x \leq 6$, $x \leq y$. If there exists such a value of $\gamma$ that it is the solution of the equation $\delta_p = \delta_q$, this value will be denoted by $\gamma_{pq}(x, y)$.

Now we are able to describe an algorithm for finding the breakpoints of the function $t_0(\gamma)$.

**Algorithm 4.1.**

1. Find the values $\gamma_{pq} \triangleq b_p/a_q$, $p = 1, 2, \ldots, n$; $q = 1, 2, \ldots, n$.
2. Find the sequence $\phi = (m_0, m_1, \ldots, m_n)$ by implementing Step 1 of Algorithm 3.1.
3. For each $p = 0, 1, \ldots, n - 1; q = 0, 1, \ldots, n - 1; p \neq q$, find the values:

\[
\gamma_{pq}(2, 2) = \frac{b_q - b_p}{(a_{m_{q+1}} - a_{m_{p+1}})}; \quad \gamma_{pq}(2, 3) = \frac{b_{q+1} - b_p + b_p}{a_{m_{q+1}}},
\]

\[
\gamma_{pq}(2, 4) = \frac{b_p}{(a_{m_{p+1}} - a_{m_{q+1}} + a_{m_{q}})}; \quad \gamma_{pq}(2, 5) = \frac{b_{q+1} + b_p}{(a_{m_{q}} + a_{m_{p+1}})},
\]

\[
\gamma_{pq}(3, 4) = \frac{b_{p+1} - b_p}{(a_{m_{q+1}} - a_{m_{p+1}})}; \quad \gamma_{pq}(3, 5) = \frac{b_{q+1} - b_{p+1} + b_p}{a_{m_{q}}},
\]

\[
\gamma_{pq}(4, 5) = \frac{b_{q+1}}{(a_{m_{p+1}} - a_{m_{p}} + a_{m_{q}})}; \quad \gamma_{pq}(5, 5) = \frac{(b_{p+1} - b_{q+1})}{(a_{m_{p}} - a_{m_{q}})}.
\]

Note that only those fractions for which both numerator and denominator are strictly positive must be calculated.

4. Form a sequence $\gamma(1), \gamma(2), \ldots, \gamma(r - 1)$ consisting of the pairwise distinct values calculated in Steps 1 and 3 sorted in nondecreasing order. Set $\gamma(0) = 0$ and $\gamma(r) = W$ where $W > \gamma(r - 1)$ is a sufficiently large number. Stop.

Note that the values $\gamma_{pq}$ found in Step 1 of Algorithm 4.1 contain also all possible values $\gamma_{pq}(1, 2), \gamma_{pq}(1, 5), \gamma_{pq}(2, 6)$ and $\gamma_{pq}(5, 6)$, Some of the values such as, e.g., $\gamma_{pq}(3, 3)$ do not exist.

It is easy to check that $r = O(n^2)$ and the running time of Algorithm 4.1 is $O(n^2 \log n)$. We now prove that Algorithm 4.1 is correct.

**Theorem 4.1.** The sequence $\Gamma = (\gamma(0), \gamma(1), \ldots, \gamma(r))$ found by Algorithm 4.1 is the sequence of all breakpoints of function $t_0(\gamma)$.

**Proof.** To prove the theorem we show that each open interval $(\gamma(j - 1), \gamma(j))$, $1 \leq j \leq r$, does not contain a breakpoint of function $t_0(\gamma)$.

We start with the breakpoints of the first type. First, notice that for any $q, q = 1, 2, \ldots, n$, the solution of equation $\gamma a_{m_q} = b_q$ belongs to the set of different values $\gamma_{pq}$ found in Step 1 of Algorithm 4.1, and, hence, to the sequence $\Gamma$. Therefore, condition (iii) is satisfied while $\gamma$ is varying in an open interval $(\gamma(j - 1), \gamma(j))$, $1 \leq j \leq r$.

The type of a city changes if $\gamma$ passes a point which is the solution of one of the equations $\gamma a_{m_q} = b_q$, $q = 1, 2, \ldots, n$, and $\gamma a_{m_{q+1}} = b_q$, $\gamma a_{m_q} - b_{q+1}, q = 1, 2, \ldots, n - 1$. Again, all solutions of these equations are found in Step 1 of Algorithm 4.1 and belong to sequence $\Gamma$.

Thus, conditions (i) and (iii) are satisfied while $\gamma$ is varying in an open interval $(\gamma(j - 1), \gamma(j))$, $1 \leq j \leq r$. We claim that condition (ii) is also satisfied. If it were violated, then for some $j$ there would exist $\tilde{\gamma} \in (\gamma(j - 1), \gamma(j))$ which would be the solution of an equation $\delta_p = \delta_q$ for some $p$ and $q$, $0 \leq p, q \leq n - 1, 0 \leq q \leq n - 1, p \neq q$. However, all possible solutions of the equations of the form $\delta_p = \delta_q$ are calculated in Step 3 of Algorithm 4.1 and, hence, they belong to sequence $\Gamma$. This implies that while $\gamma$ is varying within an interval $(\gamma(j - 1), \gamma(j))$ the $\delta$-sequence of the cities does not change, and, due to Lemma 4.1, condition (ii) is satisfied.
Thus, we have proved that an open interval \((\gamma(j-1), \gamma(j))\), \(1 \leq j \leq r\), does not contain a breakpoint of the first type.

So, when \(\gamma\) is varying within an interval \((\gamma(j-1), \gamma(j))\), \(1 \leq j \leq r\), then sequence \(\pi^*(j)\) which minimizes function \(t_0(\gamma, \pi)\) remains the same. To complete the proof, we must show that an open interval \((\gamma(j-1), \gamma(j))\), \(1 \leq j \leq r\), does not contain a breakpoint of the second type. In other words, we must show that for each optimal sequence \((i_1, i_2, \ldots, i_n)\) the signs of all expressions \(b_{ik} - \gamma a_{ik+1}, k = 1, 2, \ldots, n - 1\), do not change while \(\gamma\) is varying as long as the sequence \((i_1, i_2, \ldots, i_n)\) is optimal. But all possible breakpoints of this type belong to the sequence \(\gamma_{pq}, p = 1, 2, \ldots, n; q = 1, 2, \ldots, n\), found in Step 1 of Algorithm 4.1, and, hence, to sequence \(\Gamma\).

Having found the breakpoints of function \(t_0(\gamma)\), we may find the functions \(G_j(\gamma, w_\beta)\) of the form (2.5), \(j = 1, 2, \ldots, r\). To do this we have to calculate the values of \(\alpha(j)\) and \(\beta(j), j = 1, 2, \ldots, r\). For this purpose we may use the following.

**Algorithm 4.2.**

1. Let sequence \(\Gamma = (\gamma(0), \gamma(1), \ldots, \gamma(r))\) be found.
2. For each \(j, 1 \leq j \leq r\), do
   1. Set \(\gamma = \gamma(j)\) and implement Steps 2–9 of Algorithm 3.1 to find a sequence \(\pi^*(j) = (i_1, i_2, \ldots, i_n)\) which minimizes function \(t_0(\gamma, \pi)\) for \(\gamma \in (\gamma(j-1), \gamma(j))\).
   2. Take any \(\gamma \in (\gamma(j-1), \gamma(j))\) and find the sets
      \[N_b(j) = \{i_k | k = 1, 2, \ldots, n - 1, b_{ik} > \gamma a_{ik+1}\}\]
      \[N_a(j) = \{i_k+1 | k = 1, 2, \ldots, n - 1, b_{ik} > \gamma a_{ik+1}\}\].
   3. Set
      \[\alpha(j) = \sum_{k=1}^{n} a_{ik} - \sum_{i_k+1 \in N_a(j)} a_{ik+1}; \quad \beta(j) = \sum_{i_k \in N_b(j)} b_{ik} + b_n.\]
      Stop.

Note that for this algorithm we used the fact that it is not necessary to implement all steps of Algorithm 3.1 to find a sequence \(\pi^*(j)\) for a certain \(j\). The reason is that the sequence \(\phi\) can be considered here as having been found (see Step 2 of Algorithm 4.1), and as soon as this sequence does not depend on \(\gamma\) it can be used for the subsequent applications of Algorithm 3.1. Recall that the remaining steps of Algorithm 3.1 can be implemented in linear time. So we can conclude that the running time of Algorithm 4.2 is \(O(nr) = O(n^3)\).

One may observe that it is sufficient to look for a new sequence \(\pi^*(j)\) only if \(\gamma(j-1)\) is a breakpoint of the first type. However, this observation does not allow us
to reduce the running time of Algorithm 4.2 in the worst case, because there are \( O(n^2) \) breakpoints of this type.

5. The general algorithm

In this section we put together all the algorithms presented in the previous sections to describe the general algorithm for solving the original problem.

Algorithm 5.1 (The general algorithm).

1. Implement Algorithm 4.1 in order to find all \( r \) breakpoints of function \( t_0(\gamma) \). This requires \( O(n^2 \log n) \) time.
2. Implement Algorithm 4.2 in order to find all values \( x(j) \) and \( \beta(j), j = 1, 2, \ldots, r \). This can be done in \( O(n^3) \) time.
3. For each \( j, j = 1, 2, \ldots, r \), implement Algorithm 2.1 in order to find the values \( \gamma^*(j) \) and \( w^*_j(j) \). This requires \( O(r) = O(n^2) \) time.
4. Implement Algorithm 2.2 in order to find the desired values \( v^*_x \) and \( v^*_y \). This also requires \( O(r) = O(n^2) \) time.
5. Find the corresponding optimal schedule. This can be done in \( O(n) \) time.

It follows immediately from the previous results that the algorithm is correct. Thus, the following theorem is true.

Theorem 5.1. The two machine no wait flow shop scheduling problem with controllable machine speeds to minimize function (1.1) can be solved in \( O(n^3) \) time.

Acknowledgements

The author is grateful to C.P.M. van Hoesel, A.H.G. Rinnooy Kan and L.N. Van Wassenhove for helpful discussion. The comprehensive comments from the referees have contributed to simplifying some proofs and to improving the presentation.

References


