THE TWO-STAGE ASSEMBLY SCHEDULING PROBLEM: COMPLEXITY AND APPROXIMATION

by

Luk VAN WASSENHOVE*
Christopher N. POTTS**
S.V. SEVAST'JANOV***
V.A. STRUSEVICH****
Carin ZWANEVELD*****

N° 92/53/TM

* Professor of Operations Management and Operations Research, at INSEAD, Boulevard de Constance, Fontainebleau 77305 Cedex, France.

** Professor at the University of Southampton, U.K.

*** Professor at Institute of Mathematics, Siberian Branch of Russian Academy of Sciences, Novosibirsk, Russia.

**** Professor at Econometric Institute, Erasmus University Rotterdam, The Netherlands.

***** Professor at Tinbergen Institute, Erasmus University, Rotterdam, The Netherlands.

Printed at INSEAD,
Fontainebleau, France
THE TWO-STAGE ASSEMBLY SCHEDULING PROBLEM: COMPLEXITY AND APPROXIMATION

C.N. Potts
Faculty of Mathematical Studies,
University of Southampton, U.K.

S.V. Sevast’janov
Institute of Mathematics, Siberian Branch of
Russian Academy of Sciences, Novosibirsk, Russia

V.A. Strusevich
Econometric Institute, Erasmus University,
Rotterdam, The Netherlands

L.N. Van Wassenhove
INSEAD, Fontainebleau, France

C.M. Zwaneveld
Tinbergen Institute, Erasmus University,
Rotterdam, The Netherlands

Abstract

This paper introduces a new two-stage assembly scheduling problem. There are $m$ machines at the first stage each of which produces a component of a job. When all $m$ components are available, a single assembly machine at the second stage completes the job. The objective is to schedule jobs on the machines so that the makespan is minimized. It is shown that the search for an optimal solution may be restricted to permutation schedules. The problem is proved to be NP-hard in the strong sense even when $m = 2$. A schedule associated with an arbitrary permutation of jobs is shown to provide a worst-case ratio bound of 2, and a heuristic with a worst-case ratio bound of $2 - 1/m$ is presented. The compact vector summation technique is applied for finding approximate solutions with worst-case absolute performance guarantees.
Introduction

The two-stage assembly problem is a generalization of the two-machine flow shop problem. Informally, it can be described as follows. There are $n$ jobs to be processed. In the first stage, each of the machines $M_i$, $i = 1, 2, \ldots, m$, $m \geq 2$, processes a component of a job; these machines work independently of each other. In the second stage, the assembly machine $M_A$ assembles the $m$ prepared components of each job.

Each job $J_j$, $j = 1, 2, \ldots, n$, consists of a chain of sets of operations ($\{O_{1,j}, O_{2,j}, \ldots, O_{m,j}\}$, $O_{A,j}$). An operation $O_{i,j}$ is to be processed on machine $M_i$, $i = 1, 2, \ldots, m$, and this requires $p_{i,j}$ time. Machine $M_i$ can process at most one job at a time. An assembly operation $O_{A,j}$ is to be performed on $M_A$ and takes $p_{A,j}$ time. For any $i$ and $k$, $i = 1, 2, \ldots, m$, $k = 1, 2, \ldots, m$, $i \neq k$, operations $O_{i,j}$ and $O_{k,j}$, $j = 1, 2, \ldots, n$, are allowed to be processed simultaneously. An assembly operation $O_{A,j}$ may start only after all operations $O_{1,j}$, $O_{2,j}, \ldots, O_{m,j}$ have been completed. The assembly machine $M_A$ can assemble the components of at most one job at a time. The criterion for optimality is the makespan $C_{\text{max}}$, i.e., we need to minimize the time that all machines have completed all $n$ jobs.

The problem is frequently encountered in practice. Picture, for instance, the production of personal computers. Orders are assembled to customer specification at a packaging station. A customer typically requires a specific set of modules; a central processing unit, a hard disc, a video display unit, a printer, an appropriate keyboard, a set of manuals in the right language, etc. Although there may be only a few options for each module (e.g., there may only be five types of hard discs), a large variety of end products can still be offered to the customer by using different combinations at the packaging station. The modules are produced on independent feeder lines, say one line for the keyboards, one for the display units, etc. It is clear that this situation fits our assembly scheduling model.

Of course, there are many other situations where a set of modules are produced on independent feeder lines, followed by an assembly or a packaging step. As many industries move closer to Just-In-Time systems, this type of layout will increasingly be found. Moreover, the market pressure for larger variety combined with the need to control costs in a global competitive environment forces companies to re-design products with flat bills of materials and modular structures. It follows that the problem discussed in this paper becomes increasingly relevant.
In our analysis we assume that all jobs are simultaneously available at time zero. Zero processing times are considered as very small positive numbers. No preemption is allowed, i.e., once started, an operation cannot be interrupted before completion. We denote the two-stage assembly problem with \( m \) machines in the first stage by \( Am \mid C_{\text{max}} \).

Note that if there is only one machine in the first stage, i.e., \( m = 1 \), the two-stage assembly problem coincides with the two-machine flow shop scheduling problem to minimize the makespan. We refer to the latter problem as \( F2 \mid C_{\text{max}} \) (see, e.g., Lawler et al. (1989)).

Recall some results on the \( F2 \mid C_{\text{max}} \) problem. Let the machines be denoted by \( M_1 \) and \( M_2 \). Each job \( J_i \) consists of two operations \( O_{1i} \) and \( O_{2i} \) to be processed in this order on \( M_1 \) and \( M_2 \), respectively. Processing an operation \( O_{ij} \) takes \( p_{ij} \) time, \( i = 1, 2 \). For the \( F2 \mid C_{\text{max}} \) problem there always exists an optimal solution which is a permutation schedule, i.e., a schedule with the same job processing order on both machines. Let \( \pi = (j_1, j_2, ..., j_n) \) be an arbitrary permutation of jobs. This permutation specifies a schedule \( S \) with makespan

\[
C_{\text{max}}(S) = \max \left\{ \sum_{k=1}^{u} p_{1,j_k} + \sum_{k=u}^{n} p_{2,j_k} \right\} \text{ s.t. } u = 1, 2, ..., n.
\]

To find a permutation which specifies an optimal schedule for the \( F2 \mid C_{\text{max}} \) problem, the well-known Johnson algorithm (see Johnson (1954)) may be used. According to that algorithm, the optimal permutation starts with the jobs for which \( p_{1,i} \leq p_{2,i} \) taken in nondecreasing order of \( p_{1,i} \), followed by the rest of the jobs taken in nonincreasing order with respect to \( p_{2,i} \). Finding the optimal permutation requires \( O(n \log n) \) time.

The remainder of this paper is organized as follows. In Section 1, we show that the search for an optimal solution may be restricted to permutation schedules. Section 2 establishes that the problem is NP-hard in the strong sense even when \( m = 2 \). Approximation algorithms are proposed and analyzed in Sections 3 and 4. Section 3 presents an algorithm which has a worst-case ratio bound of \( 2 - 1/m \). In Section 4, algorithms are presented which have respective worst-case absolute bounds of \( m + 1/m \) times the largest processing time and, for \( m = 2 \), of 1.25 times the largest processing time. Some concluding remarks are contained in Section 5.
1. Permutation schedules

In this section we show that, for the $Am|C_{\text{max}}$ problem, the search for an optimal solution may be restricted to the class of permutation schedules. We also derive an analytical expression for the makespan which emphasizes close connections between the assembly problem and the two-machine flow shop scheduling problem.

We start with some notation. Let $S$ be a schedule for the $Am|C_{\text{max}}$ problem. In schedule $S$, the completion and the starting times of an operation $O_{i,j}$ are denoted by $C_{i,j}(S)$ and $R_{i,j}(S)$, respectively, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$; $C_{A,j}(S)$ and $R_{A,j}(S)$ are defined analogously. The time when a job $J_j$ is ready for assembly is denoted by $r_j(S)$. Note that $r_j(S) = \max\{C_{i,j}(S) | i = 1, 2, \ldots, m\} \leq R_{A,j}(S)$.

We note that our search for an optimal solution can be restricted to the class of schedules for which each of the first-stage machines $M_i$, $i = 1, 2, \ldots, m$, starts at time zero and has no intermediate idle time.

**Theorem 1.1.** For the $Am|C_{\text{max}}$ problem, there exists an optimal solution which is a permutation schedule.

**Proof.** Suppose there exists an optimal schedule $S$ in which the processing order for some first-stage machine $M_i$ differs from the order for the assembly machine $M_A$. In this case, there exists a pair of jobs $J_j$ and $J_k$ such that job $J_j$ is sequenced immediately before $J_k$ on $M_i$, but $J_j$ follows $J_k$ on $M_A$, perhaps with some intervening operations. We assume that in $S$ there is no idle time on $M_i$ so that $C_{i,k}(S) = C_{i,j}(S) + p_{i,k} \leq r_k(S) \leq R_{A,k}(S) \leq R_{A,j}(S)$.

Construct a new schedule $S'$ by interchanging jobs $J_j$ and $J_k$ on $M_i$. We have that $C_{i,k}(S') \leq C_{i,k}(S) \leq R_{A,k}(S')$. Besides, $C_{i,j}(S') = C_{i,k}(S) \leq R_{A,j}(S)$. Thus, in $S'$, all jobs can start on $M_A$ at the same time as they do in $S$, so schedule $S'$ is also optimal.

Repeating the same arguments shows that the processing orders on machine $M_A$ and on any machine $M_i$ can be made identical without increasing the makespan and without changing the schedule on $M_A$.\[\Box\]
In view of Theorem 1.1, we restrict our search for an optimal solution to permutation schedules. For any permutation of jobs, a corresponding schedule is constructed by processing every operation as early as possible. We now derive an expression for the makespan of such a permutation schedule.

**Theorem 1.2.** Let $\pi = (j_1, j_2, \ldots, j_n)$ be an arbitrary permutation of jobs and $S$ be the assembly schedule specified by that permutation. Then

$$C_{\text{max}}(S) = \max \left\{ \max \left\{ \sum_{i=1}^{m} p_{i,j_k} \right\}_{i=1,2,\ldots,m} + \sum_{k=1}^{n} p_{A,j_k} \right\}_{u=1,2,\ldots,n}. \tag{1.1}$$

**Proof.** Since, in $S$, all operations are processed as early as possible, we have

$$C_{A,j_1}(S) = r_{j_1}(S) + p_{A,j_1},$$

$$C_{A,j_k}(S) = \max \{r_{j_k}(S), C_{A,j_{k-1}}(S)\} + p_{A,j_k}, \quad k = 2, 3, \ldots, n,$$

from which we deduce that

$$C_{A,j_n}(S) = \max \left\{ r_{j_n}(S) + \sum_{k=1}^{n} p_{A,j_k} \right\}_{u=1,2,\ldots,n}. \quad \text{Since } C_{\text{max}}(S) = C_{A,j_n}(S) \text{ and}$$

$$r_{j_n}(S) = \max \left\{ \sum_{k=1}^{u} p_{i,j_k} \right\}_{i=1,2,\ldots,m},$$

the desired expression follows immediately.\[1\]

Note that the assembly problem corresponds to $m$ artificial two-machine flow shop scheduling problems in the following way. The $i$th flow shop consists of machine $M_i$ in the first stage and machine $M_A$ in the second stage, $i = 1, 2, \ldots, m$. Let $\pi = (j_1, j_2, \ldots, j_n)$ be an arbitrary permutation of jobs specifying schedules for the assembly problem and for each of the flow shop problems introduced above. The corresponding schedule for the $i$th flow shop is denoted by $S_i$ while, for the assembly problem, it is denoted by $S$. It follows from (1.1) that

$$C_{\text{max}}(S) = \max \{C_{\text{max}}(S_i)\}_{i=1,2,\ldots,m}. \tag{5}$$
2. Complexity

We show that the $A_{21} \parallel C_{\text{max}}$ problem is NP-hard in the strong sense. To prove this, we use the well-known 3-PARTITION problem (see, e.g., Carey and Johnson (1979)) which is NP-complete in the strong sense.

3-PARTITION. Given a set $T = \{1, 2, \ldots, 3t\}$ with a positive integer $e_i$ for each $i \in T$, and given a positive integer $E$ such that $\sum_{i \in T} e_i = tE$ and $E/4 < e_i < E/2$, can $T$ be partitioned into $t$ disjoint sets $T_1, T_2, \ldots, T_t$ such that $\sum_{i \in T_j} e_i = E$ for each $j$, $j = 1, 2, \ldots, t$?

Note that $E \geq 3$ and, if 3-PARTITION has a solution, then $|T_j| = 3$ for all $j$.

**Theorem 2.1.** The two-stage assembly problem is NP-hard in the strong sense.

**Proof.** Given an arbitrary instance of 3-PARTITION, we define the instance of the $A_{21} \parallel C_{\text{max}}$ problem with $n = 4t+1$ jobs divided into two groups: $U$-jobs denoted by $U_i$, $i = 1, 2, \ldots, 3t$, and $V$-jobs denoted by $V_j$, $j = 0, 1, \ldots, t$. We set

- $p_{1, U_i} = 3e_iE$, $p_{2, U_i} = e_i$, $p_{A, U_i} = 3e_i$, $i = 1, 2, \ldots, 3t$;
- $p_{1, V_0} = 1$, $p_{2, V_0} = 1$, $p_{A, V_0} = 3E^2$;
- $p_{1, V_j} = 3E$, $p_{2, V_j} = 3E^2 + 2E$, $p_{A, V_j} = 3E^2$, $j = 1, 2, \ldots, t-1$;
- $p_{1, V_t} = 3E$, $p_{2, V_t} = 3E^2 + 2E$, $p_{A, V_t} = 1$;
- $y = 3te^2 + 3tE + 2$.

We show that for the constructed instance of the $A_{21} \parallel C_{\text{max}}$ problem, a schedule $S_0$ with $C_{\text{max}}(S_0) \leq y$ exists if and only if 3-PARTITION has a solution.

Let $T_1, T_2, \ldots, T_t$ be a solution of 3-PARTITION. An arbitrary permutation of the $U$-jobs with indices belonging to set $T_j$ is denoted by $\pi_j(U)$. The desired schedule $S_0$ exists and is specified by the permutation $\pi_0 = (V_0, \pi_1(U), V_1, \pi_2(U), \ldots, V_{t-1}, \pi_t(U), V_t)$. It is easy to check that $C_{\text{max}}(S_0) = y$. See Fig. 2.1.

Suppose now that there exists a schedule $S_0$ with $C_{\text{max}}(S_0) \leq y$. Without loss of generality we may assume that $S_0$ is a permutation schedule, and $\pi_0$ is the corresponding permutation. Since total workload on each machine equals $y-1$, we conclude that the first job in $\pi_0$ is $V_0$, and the last job is $V_t$. Moreover, both machines $M_1$ and $M_2$ start at time zero and have
no intermediate idle time, while machine $M_A$ starts at time 1 and does not have any idle
time either. Since the jobs $V_j$, $j = 1, 2, ..., t-1$, are identical, we may assume that they
are scheduled in $S_0$ in increasing order of their numbering.

Consider the time interval $[1, 3E^2+3E+1]$. First, observe that job $V_1$ cannot start at time
1 on machine $M_2$ since this would lead to idle time on $M_A$ after processing job $V_0$. Thus,
both machines $M_1$ and $M_2$ starting from time 1 must process some of $U$-jobs and then process
job $V_1$. We denote the index set of those $U$-jobs by $T$.

If $\sum_{i \in T} e_i = E' < E$, then we get intermediate idle time on $M_A$ before processing job $V_1$. This
job is ready for the assembly operation at time $1 + 3E^2 + 2E'$ while the assembly machine
is free at time $1 + 3E^2 + 3E'$.

Similarly, if $\sum_{i \in T} e_i = E' > E$, the following argument shows that there is also
intermediate idle time on $M_A$ before processing job $V_1$. This job is not available for
processing on $M_A$ before time $1 + 3EE' + 3E = 1 + 3E^2 + 3E + 3E(E' - E)$. Also $M_A$ has exactly
$3E^2 + 3E' = 3E^2 + 3E + 3(E' - E)$ time units of processing before job $V_1$ starts. Since $E \geq
3$, there is intermediate idle time on $M_A$.

Thus, we can conclude that $\sum_{i \in T} e_i = E$. Extending these arguments, it is straightforward to
show that in each time interval $[1+3(j-1)E^2 + 3(j-1)E, 1+3(j-1)E^2 + (3j-2)E]$, $j = 1, 2, ..., t$,
machine $M_2$ processes exactly three $U$-jobs with total processing time equal to $E$. This
implies that $3$-PARTITION has a solution.
3. Heuristics with ratio performance guarantees

Since the \( Am|C_{\text{max}} \) problem is \( NP \)-hard, designing approximation algorithms is an interesting research goal. Let \( S_H \) be a schedule generated by a heuristic \( H \), while \( S^* \) is an optimal schedule. Heuristic \( H \) is said to provide the ratio performance guarantee \( \rho \) if for any problem instance \( \Delta(S_H) = C_{\text{max}}(S_H)/C_{\text{max}}(S^*) \leq \rho \).

In this section, we study the worst-case performance ratio of several heuristic algorithms. In particular, we present a heuristic with a tight worst-case ratio bound of \( 2 - 1/m \).

**Theorem 3.1.** Let \( H \) be a heuristic which generates an arbitrary permutation. Then \( \Delta(S_H) \leq 2 \) and this bound is tight.

**Proof.** Suppose that \( H \) generates permutation \( \pi = (j_1, j_2, \ldots, j_n) \). Then it follows from (1.1) that

\[
C_{\text{max}}(S_H) = \max \left\{ \max \left\{ \sum_{k=1}^{u} p_{i,j_k} \right| i = 1, 2, \ldots, m \right\} + \sum_{k=u}^{n} p_{A,j_k} \left| u = 1, 2, \ldots, n \right\} \leq \\
\leq \max \left\{ \sum_{k=1}^{n} p_{i,j_k} \right| i = 1, 2, \ldots, m \right\} + \sum_{k=1}^{n} p_{A,j_k} \leq 2C_{\text{max}}(S^*),
\]

where the last inequality is obtained from the observation that the total processing time on any machine provides a lower bound on \( C_{\text{max}}(S^*) \).

To demonstrate that this bound is tight, consider the following instance:

- \( n = 2; \)
- \( p_{i_1} = 1, i = 1, 2, \ldots, m, p_{A_1} = k; \)
- \( p_{i_2} = k, i = 1, 2, \ldots, m, p_{A_2} = 1, \)

where \( k > 1 \). It is easy to see that the permutation \((1, 2)\) is optimal and the makespan is \( k + 2 \). On the other hand, if heuristic \( H \) generates the permutation \((2, 1)\), the makespan of the corresponding schedule \( S_H \) is \( 2k + 1 \). Thus, if \( k \rightarrow \infty \), then \( \Delta(S_H) \rightarrow 2. \)

Suppose that \( \pi_i \) is a permutation which is optimal for the two-machine flow shop problem.
with machines $M_i$ and $M_A$, $i = 1, 2, ..., m$, and $\pi$ is chosen from these $m$ permutations so that the makespan for the original assembly problem is as small as possible. Then, for assembly schedule $S_H$ specified by permutation $\pi$, the bound $\Delta(S_H) \leq 2$ is still tight.

To justify this assertion, consider the following instance of the $Am1|C_{max}$ problem. There are $n = m + 1$ jobs such that

- $p_{i,j} = 1$, $i \neq j$, $p_{i,i} = k$, $i = 1, 2, ..., m$, $p_{A,j} = 1$, $j = 1, 2, ..., m$;
- $p_{i,m+1} = 2$, $p_{A,m+1} = k$, $i = 1, 2, ..., m$,

where $k > 1$. It is easily verified that an optimal assembly schedule $S^*$ is specified by the permutation $(m + 1, 1, 2, ..., m)$, and that $C_{max}(S^*) = k + m + 2$. On the other hand, each permutation $\pi_i$ is of the form $(1, 2, ..., i - 1, i + 1, ..., m, m + 1, i)$, and for the corresponding assembly schedule $S_i$ we have $C_{max}(S_i) = 2k + m + 1$. Thus, if $k \to \infty$, then $\Delta(S_i) \to 2$.

The following heuristic yields an improved worst-case ratio bound.

**Heuristic $H_0$**

1. Apply Johnson's rule to the $F2||C_{max}$ problem with the processing time of $J_j$, $j = 1, 2, ..., n$, equal to $(p_{j1} + p_{j2} + ... + p_{jm})/m$ (in the first stage) and to $p_{jA}$ (in the second stage). Let $\pi_0 = (j_1, j_2, ..., j_n)$ be the permutation found.

2. Construct the assembly schedule $S_{H_0}$ specified by the permutation $\pi_0$. Stop.

The running time of Heuristic $H_0$ is $O(nm + n \log n)$.

**Theorem 3.2.** For schedule $S_{H_0}$ found by Heuristic $H_0$

$$\Delta(S_{H_0}) \leq 2 - 1/m,$$

and this bound is tight.

**Proof.** As above, $S^*$ denotes an optimal assembly schedule. Without loss of generality, we may assume that this schedule is specified by permutation $(1, 2, ..., n)$. Then for each $i$, $i = 1, 2, ..., m$, and each $u$, $u = 1, 2, ..., n$, we have

$$C_{max}(S^*) \geq \sum_{j=1}^{u} p_{i,j} + \sum_{j=u}^{n} p_{A,j},$$
and, hence,

\[ C_{\text{max}}(S^*) \geq \sum_{i=1}^{m} \sum_{j=1}^{n} p_{i,j}/m + \sum_{u=1}^{n} p_{A,u}, \ u = 1, 2, \ldots, n. \]  

(3.2)

In turn, for permutation \( \pi_0 = (j_1, j_2, \ldots, j_n) \) found by Heuristic \( H_0 \), we have from (1.1), for some \( v, \ v = 1, 2, \ldots, n \), and for some \( h, \ h = 1, 2, \ldots, m \), that

\[ C_{\text{max}}(S_{H_0}) = \sum_{k=1}^{u} p_{h,j_k} + \sum_{k=v}^{n} p_{A,j_k}, \]

and therefore

\[ C_{\text{max}}(S_{H_0}) \leq (1 - 1/m) \sum_{k=1}^{u} p_{h,j_k} + \sum_{i=1}^{m} \sum_{k=1}^{u} p_{i,j_k}/m + \sum_{k=v}^{n} p_{A,j_k}. \]

Using the property that \( \pi_0 \) is an optimal permutation for the \( P_2 | C_{\text{max}} \) problem with processing times for job \( J_j \) equal to \((p_{j1} + p_{j2} + \ldots + p_{jm})/m\) and \( p_{A,j} \), we obtain

\[ C_{\text{max}}(S_{H_0}) \leq (1 - 1/m) \sum_{k=1}^{u} p_{h,j_k} + \max \left\{ \sum_{i=1}^{m} \sum_{k=1}^{u} p_{i,k}/m + \sum_{k=v}^{n} p_{A,k} | u = 1, 2, \ldots, n \right\}. \]

Substituting the lower bound

\[ C_{\text{max}}(S^*) \geq \sum_{k=1}^{v} p_{h,j_k} \]

and inequality (3.2), the desired bound (3.1) follows immediately.

We now prove that this bound is tight. Consider the following instance of the \( Am | C_{\text{max}} \) problem. There are \( n = (m-1)k + 1 \) jobs, \( k > m \), such that

\- \( p_{q,(q-1)k+j} = mk, \ q = 1, 2, \ldots, m - 1, \ j = 1, 2, \ldots, k; \)
\- \( p_{i,(q-1)k+j} = 1, \ i = 1, 2, \ldots, m, \ q = 1, 2, \ldots, m - 1, \ i \neq q; \ j = 1, 2, \ldots, k; \)
\- \( p_{A,(q-1)k+j} = k, \ q = 1, 2, \ldots, m - 1, \ j = 1, 2, \ldots, k; \)
\- \( p_{i,(m-1)k+1} = 1, \ i = 1, 2, \ldots, m - 1; \)
\- \( p_{m,(m-1)k+1} = mk^2, \ p_{A,(m-1)k+1} = k + 1. \)

It is easily verified that an optimal assembly schedule \( S^* \) is specified, e.g., by the permutation \( (1, k + 1, 2k + 1, \ldots, (m - 2)k + 1, 2, k + 2, 2k + 2, \ldots, (m - 2)k + 2, \ldots, k, 2k, \ldots, (m - 1)k, (m - 1)k + 1) \), and that \( C_{\text{max}}(S^*) = mk^2 + (m - 2)(k - 1) + (m - 1)k + k + 1 \). On the other hand, Heuristic \( H_0 \) generates permutation \( \pi_0 = ((m - 1)k + 1, 1, 2, \ldots, (m - 1)k) \) so that for the corresponding assembly schedule \( S_{H_0} \) we have \( C_{\text{max}}(S_{H_0}) = mk^2 + (m - 1)k^2 + k + 1 \). Thus, if \( k \to \infty \), then \( \Delta(S_{H_0}) \to 2 - 1/m \).
4. Heuristics with absolute performance guarantees

In this section, we study heuristics with another measure of their worst-case performance. A heuristic \( H \) generating schedule \( S_H \) is said to provide the absolute performance guarantee \( \alpha \) if for any problem instance \( \delta(S_H) = C_{\text{max}}(S_H) - C_{\text{max}}(S^*) \leq \alpha. \)

We present several heuristic algorithms for the \( \text{Am} | \text{C}_{\text{max}} \) problem based on some geometrical considerations.

For the \( \text{Am} | \text{C}_{\text{max}} \) problem, we denote
\[
p^* = \max \{ p_{i,j}, p_{A,j} | i = 1, 2, ..., m; j = 1, 2, ..., n \};
\]
\[
P^* = \max \left\{ \sum_{j=1}^{n} p_{i,j}, \sum_{j=1}^{n} p_{A,j} \right\} i = 1, 2, ..., m.
\]

Let \( \pi = (j_1, j_2, ..., j_n) \) be an arbitrary permutation, and \( S \) be an assembly schedule associated with that permutation. It follows from (1.1) that
\[
C_{\text{max}}(S) = \max \left\{ \max \left\{ \sum_{k=1}^{m} p_{i,j_k} \right\} i = 1, 2, ..., m \right\} + \sum_{k=1}^{m} p_{A,j_k} \leq \max \left\{ \sum_{k=1}^{m} p_{i,j_k} - \sum_{k=1}^{m} p_{A,j_k} \right\} + p^* + P^*.
\]

Thus, if one is able to find a permutation for which
\[
\sum_{k=1}^{m} p_{i,j_k} - \sum_{k=1}^{m} p_{A,j_k} \leq cp^*, i = 1, 2, ..., m, u = 1, 2, ..., n,
\]
where \( c \) is some constant, then the makespan for the corresponding assembly schedule \( S_H \) satisfies
\[
C_{\text{max}}(S_H) \leq (c + 1)p^* + P^*.
\]

Since, for an optimal assembly schedule \( S^* \) the obvious lower bound \( C_{\text{max}}(S^*) \geq P^* \) holds, that approach leads to the worst-case absolute bound \( \delta(S_H) = C_{\text{max}}(S_H) - C_{\text{max}}(S^*) \leq (c + 1)p^* \).

The problem of finding a permutation that satisfies (4.1) can be reduced to the following geometrical problem. Let \( v = (v(1), v(2), ..., v(m)) \in \mathbb{R}^m \) denote a \( m \)-dimensional vector. The length of a vector \( v \) with respect to a norm \( s \) is denoted by \( \|v\|_s \), and is called the
s-length. For \( n \) vectors \( v_1, v_2, \ldots, v_n \in \mathbb{R}^m \) and an arbitrary permutation \( \pi = (j_1, j_2, \ldots, j_n) \) of the integers 1, 2, ..., \( n \), define

\[
v^u_\pi = \sum_{k=1}^{u} v_{j_k} = (v^u_\pi(1), v^u_\pi(2), \ldots, v^u_\pi(m)), \quad u = 1, 2, \ldots, n.
\]

Given a family \( V \) of \( n \) vectors \( v_1, v_2, \ldots, v_n \in C_0 \subseteq \mathbb{R}^m \) and a body \( C \subseteq \mathbb{R}^m \), consider the problem of finding a permutation \( \pi \) such that \( v^u_\pi \in C \) for each \( u, u = 1, 2, \ldots, n \). This problem will be called the strict compact vector summation problem.

Research on this problem, which traces back to Steinitz (1913), has two angles: reducing the volume of body \( C \), and reducing the running time for finding permutation \( \pi \). See Banaszczyk (1987), Bárany (1981), Sevast'janov (1980, 1991) for results in this area.

For our purposes, the following result can be used. Let \( 0 \) denote the \( m \)-dimensional vector with zero components, and, for a body \( C \subseteq \mathbb{R}^m \) and a positive \( a \), \( aC \) denotes the set \( \{av|v \in C\} \).

**Theorem 4.1.** (see Sevast'janov (1991)) *Let \( V \) be a family of \( n \) vectors \( v_1, v_2, \ldots, v_n \in \mathbb{R}^m \) such that
\[
\sum_{j=1}^{n} v_j = 0,
\]
and each vector is contained within a convex body \( C_0 \) which is centrally symmetric, but not necessarily with respect to the origin. Then there exists a permutation \( \pi = (j_1, j_2, \ldots, j_n) \) such that
\[
v^u_\pi \in C = (m - 1 + \frac{1}{m}) C_0, \quad u = 1, 2, \ldots, n.
\]
and this permutation can be found in \( O(n^2 m^2) \) time.*

We now show how this Theorem 4.1 can be applied to the \( Am|C_{\max} \) problem. Let us assume that

\[
\sum_{j=1}^{n} p_{i,j} = \sum_{j=1}^{n} p_{A,j} = p^*, \quad i = 1, 2, \ldots, m. \tag{4.3}
\]

Construct a family \( V \) of \( n \) vectors \( v_1, v_2, \ldots, v_n \) where vector \( v_j \) which corresponds to job \( J_j \) is defined by
\[ v_j = \frac{1}{p} (p_{1,j} - p_{A,j}, p_{2,j} - p_{A,j}, \ldots, p_{m,j} - p_{A,j}), \quad j = 1, 2, \ldots, n. \]

For the vectors of \( V \), we choose either \( l_\infty \) or \( s_0 \) as a norm \( s \), where
\[
\|v\|_{l_\infty} = \max \{ \|v(i)\| \mid i = 1, 2, \ldots, m \},
\]
\[
\|v\|_{s_0} = \max \{ \|v\|_{l_\infty}, \max \{ \|v(i) - v(j)\| \mid i, j = 1, 2, \ldots, m \} \}. \]

Due to (4.3), the sum of all vectors of \( V \) is equal to 0, and each vector \( v_j \in V \) is contained in the unit ball \( C_0 = \{ v \in \mathbb{R}^m \mid \|v\|_s \leq 1 \} \). Thus, conditions of Theorem 4.1 are satisfied, and permutation \( \pi \) can be found such that (4.1) holds with \( c = (m - 1 + 1/m) \).

Then, due to (4.2), for the assembly schedule \( S_H \) associated with \( \pi \), the bound \( \delta(S_H) \leq (m + 1/m)p^* \) follows.

Note that assumption (4.3) is not crucial here. In fact, if (4.3) does not hold, we enlarge the original values of the processing times to satisfy (4.3) without changing the initial values of \( p^* \) and \( P^* \). This can be done in \( O(nm) \) time. Observe that, for an optimal schedule \( S^* \) for the perturbed problem, the lower bound \( C_{\max}(S^*) \geq P^* \) still holds. To obtain a heuristic schedule for the original problem one may keep either the starting or the completion times of all operations as in schedule \( S_H \) and restore the given processing times.

This implies the following result.

**Theorem 4.2.** For the Am1 Cmax problem, schedule \( S_H \) with \( \delta(S_H) \leq (m + 1/m)p^* \) can be found in \( O(n^2 m^2) \) time.

The Am1 Cmax problem can be related to another problem on compact vector summation. Let \( \pi = (j_1, j_2, \ldots, j_n) \) be an arbitrary permutation of jobs and \( S \) be the assembly schedule associated with \( \pi \). Due to (1.1), there exist an \( i, i = 1, 2, \ldots, m \), and a \( u, u = 1, 2, \ldots, n \), such that
\[
C_{\max}(S) = \sum_{k=1}^{u} p_{i,j_k} + \sum_{k=1}^{n} p_{A,j_k}.
\]

It follows that
\[ C_{\text{max}}(S) \leq \sum_{k=1}^{u} p_{i,j_k} - \sum_{k=1}^{u} P_{A,j_k} + P_A + P^* \leq \sum_{k=1}^{u} p_{i,j_k} - \sum_{k=1}^{u} P_{A,j_k} + P^* + P^* \]

and that

\[ C_{\text{max}}(S) \leq \sum_{k=1}^{u-1} p_{i,j_k} - \sum_{k=1}^{u-1} P_{A,j_k} + p_{i,j} + P^* \leq \sum_{k=1}^{u-1} p_{i,j_k} - \sum_{k=1}^{u-1} P_{A,j_k} + P^* + P^*. \]

This implies

\[ C_{\text{max}}(S) \leq \min \left\{ \sum_{k=1}^{u-1} p_{i,j_k} - \sum_{k=1}^{u-1} P_{A,j_k}, \sum_{k=1}^{u} p_{i,j_k} - \sum_{k=1}^{u} P_{A,j_k} \right\} + P^* + P^*. \]

Thus, if one is able to find a permutation \( \pi \) such that

\[ \min \left\{ \sum_{k=1}^{u-1} p_{i,j_k} - \sum_{k=1}^{u-1} P_{A,j_k}, \sum_{k=1}^{u} p_{i,j_k} - \sum_{k=1}^{u} P_{A,j_k} \right\} \leq c P^*, \quad (4.4) \]

where \( c \) is some constant, then the makespan for the corresponding assembly schedule \( S_H \) satisfies (4.2) which, in turn, implies \( \delta(S_H) \leq (c + 1)p^* \).

A geometrical problem related to that of finding permutation \( \pi \) satisfying (4.4) can be formulated as follows. Given a family \( V \) of \( n \) vectors \( v_1, v_2, \ldots, v_n \in C_0 \subset \mathbb{R}^m \) and a body \( C \subset \mathbb{R}^m \), consider the problem of finding a permutation \( \pi \) such that either \( v_{\pi}^{u-1} \in C \) or \( v_{\pi}^u \in C \) for each \( u, u = 1, 2, \ldots, n \). This problem will be called the semi-strict compact vector summation problem with respect to \( C \).

We concentrate on a special case of that problem with \( m = 2 \) and \( s = s_0 \). Let \( V \) be a family of \( n \) two-dimensional vectors \( v_1, v_2, \ldots, v_n \) such that \( \| v_j \| \leq 1 \) and \( \sum_{j=1}^{n} v_j = 0 \). Consider the semi-strict compact vector summation problem with respect to the region \( C_{a,b} = \{ v = (v(1), v(2)) \in \mathbb{R}^2 \mid v(1) \leq a, v(2) \leq b \}, \ a > 0, \ b > 0 \).

Without loss of generality, we assume that \( V \) contains no zero vector: zero vectors can always be arbitrarily inserted in the resulting permutation.
In what follows it is also assumed that the vectors of family $V$ are numbered in nondecreasing order of the arguments of the complex numbers $-v_j(2) - iv_j(1)$ where $i = \sqrt{-1}$. This numbering is shown in Fig. 4.1. That figure also presents the unit ball of norm $s_0$. 

Let $\nu$ be the identity permutation $(1, 2, \ldots, n)$. We note that $\nu$ contains four subsequences, some of which may be empty: $(1, 2, \ldots, f)$ corresponds to vectors $\nu = (\nu(1), \nu(2))$ where $\nu(1) \leq 0$ and $\nu(2) < 0$; $(f + 1, f + 2, \ldots, g)$ corresponds to vectors $\nu$ where $\nu(1) < 0$ and $\nu(2) \geq 0$; $(g + 1, g + 2, \ldots, h)$ corresponds to vectors $\nu$ where $\nu(1) \geq 0$ and $\nu(2) > 0$; and $(h + 1, h + 2, \ldots, n)$ corresponds to vectors $\nu$ where $\nu(1) > 0$ and $\nu(2) \leq 0$. Thus, the vectors taken in the sequence $\nu$ can be placed on the plane to constitute the convex polygon $M$ (see Fig. 4.2). Note that the endpoints of partial sums $\nu^f, \nu^g$, and $\nu^h$ correspond to left-most, lowest, and right-most vertices of polygon $M$, respectively.
We describe an algorithm which solves the semi-strict compact vector summation problem with respect to the region $C_{a,b}$, provided that

$$\sqrt{a} + \sqrt{b} \geq 1. \quad (4.5)$$

The algorithm scans the vectors of family $V$ in the sequence $\nu$ and finds a desired permutation $\pi = (j_1, j_2, \ldots, j_n)$.

**Algorithm 4.1**

1. If the subsequence $(1, 2, \ldots, f)$ of permutation $\nu$ is empty, set $u = 0$, $v_{\nu}^u = 0$, and go to Step 2. Otherwise, set $j_k = k$, $k = 1, 2, \ldots, f$; $u = f$; $v_{\nu}^u = v_{\nu}^f$.

2. Let $(s, s + 1, \ldots, t - 1, t)$ be a subsequence of permutation $\nu$ whose elements are not yet included in the permutation $\pi$.

   If $v_{\nu}^u + v_s \in C_{a,b}$, then set $j_{u+1} = s$, $v_{\pi}^{u+1} = v_{\pi}^u + v_s$, $u = u + 1$, and go to Step 3.

   If $v_{\nu}^u + v_t \in C_{a,b}$, then set $j_{u+1} = t$, $v_{\pi}^{u+1} = v_{\pi}^u + v_t$, $u = u + 1$, and go to Step 3.

   Otherwise, set $j_{u+1} = s$, $j_{u+2} = t$, $v_{\pi}^{u+2} = v_{\pi}^u + v_s + v_t$, $u = u + 2$, and go to Step 3.

3. If $u \leq n$ go to Step 2, otherwise stop.

Suppose that $u$ iterations of Algorithm 4.1 are made so that $v_{\pi}^u \in C_{a,b}$, and the vectors $v_s, v_{s+1}, \ldots, v_t$ are not included into the current partial sum $v_{\pi}^u$. Before proving that the algorithm generates a permutation with the required properties, we make some preliminary remarks.

**Remark 1.** If $v_s(1) > 0$, then $v_{j}(1) \geq 0$, for all $j = s + 1, \ldots, t$, and, since $\sum_{j=1}^{n} v_j(1) = 0$, it follows that $v_{\pi}^u(1) + v_s(1) \leq 0$. Similarly, if $v_s(2) > 0$ and $v_t(2) > 0$, then $v_j(2) \geq 0$, for all $j = s + 1, \ldots, t - 1$, and, therefore, $v_{\pi}^u(2) + v_t(2) \leq 0$ and $v_{\pi}^u(2) + v_s(2) \leq 0$.

For a vector $x = (x(1), x(2)) \in \mathbb{R}^2$, consider the straight line $T(x) = \{tx | t \in \mathbb{R}\}$. Two open halfplanes $L^0(x)$ and $R^0(x)$ can be specified consisting of the points located on the left and on the right from the line $T(x)$, respectively, if to move along $T(x)$ in the direction of $x$. Analytically, these halfplanes are defined as

$$L^0(x) = \{y = (y(1), y(2)) \in \mathbb{R}^2 | x(1)y(2) - x(2)y(1) < 0\},$$

$$R^0(x) = \{y = (y(1), y(2)) \in \mathbb{R}^2 | x(1)y(2) - x(2)y(1) > 0\}.$$
\[ R^0(x) = \{ y = (y(1), y(2)) \in \mathbb{R}^2 | x(1)y(2) - x(2)y(1) > 0 \} \]

Closed left and right halfplanes \( L(x) \) and \( R(x) \) are defined analogously.

**Remark 2.** The vector corresponding to the partial sum \( v^u_\pi + v_a \) is a diagonal of the polygon \( M \), and the sides of \( M \) corresponding to the vectors \( v_{s+1}, v_{t-1}, \ldots, v_t \) are located on the left from that diagonal (see Fig. 4.2). Thus, if the vectors \( v^u_\pi + v_a \) and \( v_t \) are drawn starting from the origin, vector \( v_t \) is on the right from vector \( v^u_\pi + v_a \), i.e., \( v_t \in R(v^u_\pi + v_a) \). Similarly, it may be observed that \( v_s \in L(v^u_\pi + v_t) \).

**Theorem 4.3.** For any family \( V \) of \( n \) two-dimensional vectors \( v_1, v_2, \ldots, v_n \) such that \( \sum_{j=1}^{n} v_j = 0 \), and for any positive \( a \) and \( b \) satisfying (4.5) the semi-strict summation problem with respect to the region \( C_{a,b} \) can be solved in \( O(n \log n) \) time by Algorithm 4.1.

**Proof.** First, note that finding permutation \( \nu \) requires \( O(n \log n) \) time, while Algorithm 4.1 takes only \( O(n) \) time.

Suppose that \( u \) iterations of Algorithm 4.1 are made so that \( v^u_\pi \in C_{a,b} \) and the vectors \( v_a, v_{s+1}, \ldots, v_{t-1}, v_t \) are not included into the found partial sum \( v^u_\pi \). To prove the correctness of Algorithm 4.1 it suffices to show that if \( v^u_\pi + v_a \notin C_{a,b} \) and \( v^u_\pi + v_t \notin C_{a,b} \), then \( v^u_\pi + v_a + v_t \in C_{a,b} \).

Let us define
\[ C_1 = \{ v = (v(1), v(2)) | v(1) > a, v(2) \leq b \} \],
\[ C_1 = \{ \mathbf{v} = (v(1), v(2)) | v(1) > a, v(2) \leq 0 \}, \]
\[ C_2 = \{ \mathbf{v} = (v(1), v(2)) | v(1) \leq a, v(2) > b \}, \]
\[ C_2' = \{ \mathbf{v} = (v(1), v(2)) | v(1) \leq 0, v(2) > b \}. \]

The defined regions are shown in Fig. 4.3.

Since \( \mathbf{v}_u^\pi \in C_{a,b} \), it follows from Remark 1 that if \( \mathbf{v}_u^\pi + \mathbf{v}_s \notin C_{a,b} \), then
\[
\mathbf{v}_u^\pi + \mathbf{v}_s \in C_2, \tag{4.6}
\]
and, therefore
\[
v_s(2) > 0. \tag{4.7}
\]
Similarly, if \( \mathbf{v}_u^\pi + \mathbf{v}_s \notin C_{a,b} \) and \( \mathbf{v}_u^\pi + \mathbf{v}_t \notin C_{a,b} \), then
\[
\mathbf{v}_u^\pi + \mathbf{v}_t \in C_1
\]
and
\[
v_t(1) > 0. \tag{4.8}
\]
Moreover, if \( v_s(2) > 0 \), then (4.7) and Remark 1 imply that \( v_u^\pi(2) + v_s(2) < 0 \) which contradicts (4.6). Thus,
\[
v_t(2) \leq 0. \tag{4.9}
\]
Similarly, it can be shown that
\[
v_s(1) \leq 0.
\]
For two vectors \( \mathbf{x} \) and \( \mathbf{y} \), it is obvious that, if \( \mathbf{y} \in L(\mathbf{x}) \) (and \( \mathbf{x} \in R(\mathbf{y}) \)), then \( \mathbf{x} + \mathbf{y} \in L(\mathbf{x}) \) and \( \mathbf{x} + \mathbf{y} \in R(\mathbf{y}) \). Therefore, due to Remark 2, the relations
\[
\mathbf{v}_u^\pi + \mathbf{v}_s + \mathbf{v}_t \in R(\mathbf{v}_u^\pi + \mathbf{v}_s), \tag{4.10}
\]
and
\[
\mathbf{v}_u^\pi + \mathbf{v}_s + \mathbf{v}_t \in L(\mathbf{v}_u^\pi + \mathbf{v}_t)
\]
hold.

We now prove that
\[
\mathbf{v}_u^\pi + \mathbf{v}_s \in C_2'. \tag{4.11}
\]
Suppose that (4.11) does not hold. Then, due to (4.6), \( \mathbf{x} = \mathbf{v}_u^\pi + \mathbf{v}_s \in C_2 \setminus C_2' \), and \( x(1) > 0 \),
This implies that the set \( \{ y = (y(1), y(2)) \in \mathbb{R}^2 \mid y(1) > 0, y(2) \leq 0 \} \) belongs to the halfplane \( L^0(x) \). In turn, due to (4.8) and (4.9), this means that \( v_t \in L^0(x) \), and, hence, \( x + v_t \in L^0(x) \). The latter contradicts (4.10).

Using a symmetric argument, it can be proved that

\[
v_s^u + v_t \in C_1.
\]

For a vector \( v = (v(1), v(2)) \in \mathbb{R}^2 \) such that \( v(1) < 0 \) and \( v(2) > 0 \), the \( s_0 \)-length of the segment of the straight \( \{ tv \mid t \in \mathbb{R} \} \) located within the region \( C_{a,b} \) can be defined as

\[
l(v) = \max \{ |a - \gamma b|, |a/\gamma - b|, |(a - \gamma b) - (a/\gamma - b)| \},
\]

where \( \gamma = v(1)/v(2) \). Since \( \gamma < 0 \), it follows that \( l = a + |\gamma|b + a/|\gamma| + b \). It is evident that the function \( f(x) = a/x + bx, a > 0, b > 0, x > 0 \), reaches its minimum at \( x = \sqrt{a/b} \), which implies that \( a/|\gamma| + b|\gamma| \geq 2\sqrt{ab} \). Then it immediately follows that

\[
l(v) \geq \left[ \sqrt{a} + \sqrt{b} \right]^2. \tag{4.12}
\]

We are now able to complete the proof of the theorem by showing that if \( v_s^u + v_s \notin C_{a,b} \) and \( v_s^u + v_t \notin C_{a,b} \), then \( v_s^u + v_s + v_t \in C_{a,b} \). To do this, for the vector \( y = v_s^u + v_s + v_t \), we prove that \( y(1) \leq a \). The proof of the inequality \( y(2) \leq b \) is symmetric.

Let \( x = v_s^u + v_s \). It follows from (4.10) that \( y \in R(x) \), and, hence, due to (4.11), \( y \notin C_1/C_1^\prime \). To prove that \( y(1) \leq a \) we need to show that \( y \notin C_1^\prime \).

---

**Fig. 4.4.**

---

19
Suppose that \( y \in C_i \). Connect the points \((y(1), y(2))\) and \((x(1), x(2))\) with the segment \( D \). Note that the \( s_0 \)-length of \( D \) equals \( \|v_t\|_{s_0} \). Since \( y \in R(x) \), when moving along \( D \) from \((y(1), y(2))\) to \((x(1), x(2))\), the origin remains on the left. This implies that the \( s_0 \)-length of \( D \) exceeds the \( s_0 \)-length \( l(x) \) of the segment of the straight line \( \{tx \mid t \in \mathbb{R}\} \) located within the region \( C_{a,b} \) (see Fig. 4.4). Due to (4.5) and (4.12), we obtain \( l(x) \geq 1 \). Thus, \( \|v_t\|_{s_0} > 1 \) which is impossible. 

We now show how Theorem 4.3 can be applied to the \( A2\|C_{\max} \) problem.

**Theorem 4.4.** For the \( A2\|C_{\max} \) problem, a schedule \( S_H \) such that

\[
C_{\max}(S_H) \leq 1.25p^* + P^*,
\]

and \( \delta(S_H) \leq 1.25p^* \) can be found in \( O(n\log n) \) time. The coefficient 1.25 in (4.13) cannot be reduced.

**Proof.** Without loss of generality, assume that the \( A2\|C_{\max} \) problem satisfies (4.3). Introduce a family \( V \) of \( n \) two-dimensional vectors \( v_1, v_2, \ldots, v_n \) such that

\[
v_j = \frac{1}{p^*}(p_{1j} - p_{A_j}, p_{2j} - p_{A_j}), \quad j = 1, 2, \ldots, n.
\]

Note that \( \|v_j\|_{s_0} \leq 1 \). If one applies Algorithm 4.1 to family \( V \) to solve the semi-strict summation problem with respect to the region \( C_{a,b} \) with \( a = b = 0.25 \), one obtains permutation \( \pi \) such that

\[
\min \{v_\pi^{-1}(i), v_\pi(i)\} \leq 0.25, \quad u = 1, 2, \ldots, n, \quad i = 1, 2,
\]

which implies that (4.4) holds with \( c = 0.25 \). Thus, for schedule \( S_H \) associated with \( \pi \), we obtain (4.13), and, therefore, \( \delta(S_H) \leq 1.25p^* \).

To complete the proof, we show that, for any \( \varepsilon > 0 \), there exists an instance of the \( A2\|C_{\max} \) problem such that \( C_{\max}(S) > (1.25 - \varepsilon)p^* + P^* \) for any schedule \( S \).

Given an \( \varepsilon > 0 \), take an odd \( n \geq 1/\varepsilon \) and construct the following instance of the \( A2\|C_{\max} \) problem. There are \( n + 1 \) jobs. The jobs \( J_1, J_2, \ldots, J_n \), called the \( U \)-jobs, are identical and their processing times on machines \( M_1, M_2, \) and \( M_A \) are equal to 1, \( 1 - 1/n \), and \( 1 - 1/(2n) \), respectively. The processing times of job \( J_0 \) are equal to 0, 1, and \( 1/2 \), respectively. Note that the workload of each machine equals \( n \), i.e., \( P^* = n \), while the
largest processing time \( p^* = 1 \). We show that for any schedule the total idle time on \( M_A \) exceeds \( 1.25 - \varepsilon \).

If job \( J_0 \) is processed first, then the total idle time on \( M_A \) before starting the next \( U \)-job equals \( 1.5 - 1/n > 1.25 - \varepsilon \).

For a \( k \leq n - 1 \), consider a permutation of the jobs such that exactly \( k \) of the \( U \)-jobs precede job \( J_0 \). The total idle time on \( M_A \) before processing job \( J_0 \) equals \( 1 + (k-1)/(2n) \). For \( k \leq (n - 1)/2 \), this idle time is at least \( 1.25 - 0.75/n > 1.25 - \varepsilon \). On the other hand, for \( k \geq (n - 1)/2 \), the total idle time on \( M_A \) before the first \( U \)-job after \( J_0 \) starts on that machine equals \( 1.5 - (k+2)/(2n) \). We deduce that the idle time on \( M_A \) is at least \( 1.25 - 0.75/n > 1.25 - \varepsilon \) for \( k \geq (n - 1)/2 \) as well.

This completes the proof of the theorem. \( \blacksquare \)

5. Concluding remarks

This paper establishes the complexity status of the two-stage assembly scheduling problem to minimize the makespan. Several heuristic algorithms are presented, accompanied by the worst-case analysis of their performance. Of special interest is the use of geometrical ideas that arise in compact vector summation problems to obtain absolute performance guarantees. Although there is some literature on these methods (most of which is in Russian), they have not been widely used in scheduling.

Since the assembly scheduling model has many practical applications, it is desirable to design enumerative methods based, for example, on the branch-and-bound ideas. Another direction of further research is developing approximation algorithms with better worst-case performance guarantees.

REFERENCES


