Chapter 2
Pairwise Interchange Argument and Priority Rules

In this chapter, we consider a group of methods that are applicable to problems of optimizing functions over a set of all permutations. The correctness of the basic methods is typically proved by the pairwise interchange argument, and the basic methods can be further extended to become applicable to solve more general problems.

In this chapter and throughout the book, for job $j = \pi(r)$ that occupies the rth position in the sequence $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ of jobs of set $N = \{1, 2, \ldots, n\}$, the completion time in a schedule S that processes the jobs on a single machine in accordance with permutation π is denoted either by $C_j(S)$ or by $C_{\pi(r)}$. The corresponding value can be written as

$$C_{\pi(r)} = \sum_{u=1}^{r} p_{\pi(u)}, \quad 1 \le r \le n. \tag{2.1}$$

If a schedule S is determined by a permutation π, instead of writing the objective function as $F(S) = \sum C_j(S)$, we may write $F(\pi) = \sum C_{\pi(r)}$; the same is applied to other functions, e.g., to the weighted sum of the completion times $\sum w_j C_j(S)$.

In Sect. 2.1, we give a proof of the classical result by Hardy et al. (1934) on minimizing a linear form over a set of permutations and show its implications for various scheduling problems, including problem $1||\sum C_j$. The proofs provided are based on the so-called pairwise interchange argument, which is an often used tool in this book. We also introduce important concepts of a priority rule and of a 1-priority that are widely used throughout this book. In Sect. 2.2, a priority rule is derived for problem $1||\sum w_j C_j$ of minimizing the weighted sum of the completion times on a single machine. In Sect. 2.3, the obtained results are extended to problems $Pm||\sum C_j$ and $Qm||\sum C_j$ of minimizing total completion time on parallel (identical or uniform) machines.

V.A. Strusevich and K. Rustogi, *Scheduling with Times-Changing Effects and Rate-Modifying Activities*, International Series in Operations Research & Management Science 243, DOI 10.1007/978-3-319-39574-6_2

2.1 Minimizing a Linear Form

Given two arrays $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, and two arbitrary permutations $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ and $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$, consider an expression

$$L(\pi, \sigma) = \sum_{j=1}^{n} a_{\pi(j)} b_{\sigma(j)} \tag{2.2}$$

that is often called a *linear form*. The value $L(\pi, \sigma)$ is the sum of products of elements, with one element from array **a** and the other from array **b**. Introduce the problem of minimizing a linear form, i.e., the problem of finding permutations $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ and $\psi = (\psi(1), \psi(2), \ldots, \psi(n))$ such that

$$L(\varphi, \psi) \leq L(\pi, \sigma).$$

The problem admits a natural graph theoretical interpretation. Let $G = (U, V; E)$ be an undirected complete bipartite graph. The set of vertices consists of two parts $U = \{u_1, u_2, \ldots, u_n\}$ and $V = \{v_1, v_2, \ldots, v_n\}$. The set E of its edges consists of n^2 edges (u_i, v_j), where vertex $u_i \in U$ is associated with a number a_i, while vertex $v_j \in V$ is associated with a number b_j. The weight of an edge $(u_i, v_j) \in E$ is defined as the product $a_i b_j$. A collection $M \subset E$ of n edges is called a (perfect) *matching*, if no two edges in M are adjacent to the same vertex. The matching defined by given permutations π and σ is the set of edges $\{(u_{\pi(j)}, v_{\sigma(j)}) | 1 \leq j \leq n\}$. The weight of a matching M is defined as the sum of all weights of edges in M. The problem of minimizing the linear form $L(\pi, \sigma)$ is equivalent to finding a matching of the smallest total weight. If $(u_i, v_j) \in M$, we say that the values a_i and b_j *match*.

Example 2.1 Figure 2.1 shows a complete bipartite graph $G = (U, V; E)$ with three vertices in each part. Let the values of (a_1, a_2, a_3) and (b_1, b_2, b_3) that are associated with the vertices of set U and set V, respectively, be as given in Table 2.1. Given permutations $\pi_1 = (1, 2, 3)$ and $\sigma_1 = (1, 2, 3)$, the matching M_1 consists of the edges (u_1, v_1), (u_2, v_2), and (u_3, v_3), so that the value of the linear form $L(\pi, \sigma)$ can be computed as

$$L(\pi_1, \sigma_1) = 5 \times 2 + 2 \times 1 + 4 \times 3 = 24,$$

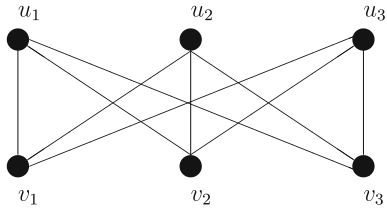

Fig. 2.1 A complete bipartite graph $G = (U, V; E)$ for Example 2.1

Table 2.1 The arrays for Example 2.1

j	1	2	3
a_j	5	2	4
b_j	2	1	3

while for the permutations $\pi_2 = (3, 1, 2)$ and $\sigma_2 = (1, 2, 3)$, the matching M_2 consists of the edges (u_3, v_1), (u_1, v_2), and (u_2, v_3), so that the value of the linear form $L(\pi, \sigma)$ can be computed as

$$L(\pi_2, \sigma_2) = 4 \times 2 + 5 \times 1 + 2 \times 3 = 19.$$

The problem of minimizing a linear form can be simplified, so that only one permutation is needed, e.g., φ, that is defined with respect to a chosen permutation ψ. In particular, we may take a permutation ψ such that the sequence $\left(b_{\psi(j)} | 1 \leq j \leq n\right)$ is non-increasing. It is convenient to assume that the components of array **b** are renumbered in such a way that

$$b_1 \geq b_2 \geq \cdots \geq b_n. \tag{2.3}$$

Under this numbering, a linear form can be written as

$$L(\pi) = \sum_{j=1}^{n} a_{\pi(j)} b_j, \tag{2.4}$$

and to minimize $L(\pi)$, we need to find a permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ of the components of array **a**, such that the inequality

$$L(\varphi) = \sum_{j=1}^{n} a_{\varphi(j)} b_j \leq L(\pi) = \sum_{j=1}^{n} a_{\pi(j)} b_j \tag{2.5}$$

holds for any permutation π.

It is clear intuitively that $L(\pi)$ will take smaller values if larger values of b_j are matched to smaller values of a_j. The statement below formalizes this observation.

Theorem 2.1 *Provided that (2.3) holds, permutation* $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ *such that*

$$a_{\varphi(1)} \leq a_{\varphi(2)} \leq \cdots \leq a_{\varphi(n)} \tag{2.6}$$

satisfies (2.5), i.e., minimizes the linear form $L(\pi)$.

Proof The proof given below utilizes the so-called *pairwise interchange argument*. Assuming that there exists an optimal sequence that does not satisfy a certain rule, a permutation that delivers a smaller value of the objective function can be obtained by interchanging the two adjacent elements that do not obey that rule.

In our case, suppose that $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ is a permutation that satisfies (2.5), but does not follow the rule (2.6). Then, there exists an index $k < n$ such that

$$a_{\varphi(1)} \le \cdots \le a_{\varphi(k-1)} \le a_{\varphi(k)};\ a_{\varphi(k)} > a_{\varphi(k+1)}. \tag{2.7}$$

Consider the permutation $\varphi' = (\varphi'(1), \varphi'(2), \ldots, \varphi'(n))$, obtained from φ by swapping the elements $\varphi(k)$ and $\varphi(k+1)$, i.e.,

$$\begin{aligned} &\varphi'(j) = \varphi(j),\ 1 \le j \le k-1;\ k+1 \le j \le n; \\ &\varphi'(k) = \varphi(k+1),\ \varphi'(k+1) = \varphi(k). \end{aligned} \tag{2.8}$$

Define $\Delta := L(\varphi) - L\big(\varphi'\big)$. Due to optimality of φ, we must have $\Delta \le 0$. However,

$$\begin{aligned} \Delta &= \left(\sum_{j=1}^{k-1} a_{\varphi(j)}b_j + a_{\varphi(k)}b_k + a_{\varphi(k+1)}b_{k+1} + \sum_{j=k+1}^{n} a_{\varphi(j)}b_j\right) \\ &\quad - \left(\sum_{j=1}^{k-1} a_{\varphi(j)}b_j + a_{\varphi(k+1)}b_k + a_{\varphi(k)}b_{k+1} + \sum_{j=k+1}^{n} a_{\varphi(j)}b_j\right) \\ &= \big(a_{\varphi(k)}b_k + a_{\varphi(k+1)}b_{k+1}\big) - \big(a_{\varphi(k+1)}b_k + a_{\varphi(k)}b_{k+1}\big) \\ &= b_k\big(a_{\varphi(k)} - a_{\varphi(k+1)}\big) - b_{k+1}\big(a_{\varphi(k)} - a_{\varphi(k+1)}\big) = (b_k - b_{k+1})\big(a_{\varphi(k)} - a_{\varphi(k+1)}\big). \end{aligned}$$

Since $b_k \ge b_{k+1}$ due to (2.3) and $a_{\varphi(k)} > a_{\varphi(k+1)}$ due to (2.7), we deduce that $\Delta > 0$.

Thus, permutation φ cannot be a solution that minimizes the linear form $L(\pi)$. Repeating this argument as many times as required, we conclude that for an optimal permutation φ, the condition (2.6) must hold. □

Thus, we can describe the following algorithm for solving the problem of minimizing a linear form. The permutation found by the algorithm matches larger components of array **b** to smaller components of array **a**.

Algorithm Match

INPUT: Two (unsorted) arrays $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$
OUTPUT: A permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ that satisfies (2.5)

Step 1. If required, renumber the components of array **b** so that (2.3) holds.
Step 2. Output a permutation φ such that (2.6) holds.

Algorithm Match reduces to two sorting procedures and therefore requires $O(n \log n)$ time. Simple as it is, the algorithm still plays an important role in optimization over permutations, including numerous scheduling applications discussed in this book.

Applying Algorithm Match to the data in Example 2.1, we first renumber the items so that (2.3) holds, i.e., item 3 with the largest b-value becomes item 1, item 1 becomes item 2, and item 2 becomes item 3. With respect to this new numbering,

$$b_1 = 3 > b_2 = 2 > b_3 = 1,$$

and Algorithm Match outputs permutation $\varphi = (3, 1, 2)$, so that

$$a_{\varphi(1)} = a_3 = 2 < a_{\varphi(2)} = a_1 = 4 < a_{\varphi(3)} = a_2 = 5,$$

and the minimum value of the linear form (the smallest weight of a matching in graph shown in Fig. 2.1) is equal to

$$2 \times 3 + 4 \times 2 + 5 \times 1 = 19,$$

i.e., smaller values in one array are matched to larger values in the other array.

2.1.1 Minimizing Total Completion Time on a Single Machine

Consider a single machine problem, in which the jobs of set $N = \{1, 2, \ldots, n\}$ have to be processed on a single machine. The processing time of job $j \in N$ is equal to p_j. It is required to minimize total completion time, i.e., the sum of the completion times. According to the three-field scheduling notation described in Sect. 1.2, the problem can be denoted by $1||\sum C_j$. Clearly, an optimal schedule for the problem is defined by a permutation of jobs.

Suppose that the jobs are sequenced in accordance with some permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. In accordance with (2.1), the total completion time can be written as

$$F(\pi) = \sum_{j=1}^{n} C_{\pi(j)} = \sum_{j=1}^{n} \sum_{i=1}^{j} p_{\pi(i)} = p_{\pi(1)} + \big(p_{\pi(1)} + p_{\pi(2)}\big) + \big(p_{\pi(1)} + p_{\pi(2)} + p_{\pi(3)}\big)$$
$$+ \cdots + \big(p_{\pi(1)} + p_{\pi(2)} + \cdots + p_{\pi(n-1)}\big) + \big(p_{\pi(1)} + p_{\pi(2)} + \cdots + p_{\pi(n-1)} + p_{\pi(n)}\big).$$

It can be seen that the contribution of job $\pi(1)$ to the objective function is $np_{\pi(1)}$, that of job $\pi(2)$ is $(n-1)p_{\pi(1)}$, etc., so that the jth job in the sequence contributes $(n-j+1)p_{\pi(j)}$, $1 \le j \le n$. Thus, we can rewrite

$$F(\pi) = \sum_{j=1}^{n} (n - j + 1) p_{\pi(j)}. \tag{2.9}$$

Clearly, $F(\pi)$ is a linear form similar to (2.4) with

$$a_j = p_j,\ b_j = (n - j + 1),\ 1 \le j \le n,$$

so that the b-values form the decreasing sequence $(n, n-1, \ldots, 1)$. As follows from Theorem 2.1, a permutation φ that minimizes $F(\pi)$ can be found by sequencing the jobs in non-decreasing order of their processing times.

This means that function $F(\pi)$ of the form (2.9) can be minimized by applying a priority rule. This is very important concept that is widely used in this book.

Definition 2.1 For a scheduling problem of sequencing the jobs of set $N = \{1, 2, \ldots, n\}$, an objective function $\Phi(\pi)$ to be minimized over a set of permutations is said to admit a solution by a *priority rule* if job j can be associated with a priority (or, more precisely, a 1-*priority*) $\omega(j)$, and an optimal permutation can be found by sorting the jobs in accordance with these 1-priorities.

Unless confusion arises, we will normally refer to the values of $\omega(j)$ as 1-priorities and not priorities. The term "1-priority" stresses that the values $\omega(j)$ are associated with individual jobs. We will use the term "priority" for problems with precedence constraints (see, e.g., Chap. 3), where priorities will be associated with subsequences of jobs, rather than with individual jobs.

Two priority rules are often used in scheduling and are of special importance in this book.

Definition 2.2 The jobs of set $N = \{1, 2, \ldots, n\}$ are said to follow the *SPT* rule (*shortest processing time*) if the jobs are renumbered so that

$$p_1 \le p_2 \le \cdots \le p_n, \tag{2.10}$$

and to follow the *LPT* rule (*longest processing time*) if the jobs are renumbered so that

$$p_1 \ge p_2 \ge \cdots \ge p_n. \tag{2.11}$$

Thus, the following has been proved.

Theorem 2.2 *For problem* $1||\sum C_j$, *an optimal permutation can be found in* $O(n \log n)$ *time, by sequencing the jobs in accordance with the SPT rule.*

As a rule, 1-priorities are defined in such a way that an optimal permutation is found by sorting the jobs in *non-increasing* order of $\omega(j)$, i.e., the higher the 1-priority is, the earlier the corresponding job is scheduled. For problem $1||\sum C_j$, its 1-priorities are defined either as $\omega(j) := -p_j$ or as $\omega(j) := 1/p_j$, $j \in N$.

Notice that the SPT sequence of jobs solves problem $1||\sum C_j$, provided that the resulting permutation is formed from front to rear. It is also possible to fill an optimal permutation from rear to front, so that the job with the largest processing time is sequenced in the last position, the one with the second largest processing time takes

the position one before last, etc. This can be interpreted as follows: Scan the jobs in accordance with the LPT sequence and assign the next job to the right most available position of the resulting permutation.

The theorem below demonstrates an important role of the SPT rule for minimizing more general functions than the total completion time $\sum C_j$, such as $\sum C_j^z$ where z is a given positive number, and $\xi C_{\max} + \eta \sum C_j^z$, where ξ and η are positive coefficients.

Theorem 2.3 *Let $\pi = (\pi(1), \ldots, \pi(n))$ be a permutation, in which two jobs u and v such that*

$$p_u > p_v \tag{2.12}$$

occupy two consecutive positions r and $r+1$, i.e., $u = \pi(r)$ and $v = \pi(r+1)$. Let permutation π' be obtained from π by swapping the jobs u and v. For a single machine problem, let $C_{\pi(h)}$ and $C_{\pi'(h)}$ denote the completion time of the job sequenced in the hth position in permutation π and π', respectively, $1 \le h \le n$. Then, the equalities

$$C_{\pi(h)} - C_{\pi'(h)} = \begin{cases} 0, & \text{for } 1 \le h \le n,\ h \ne r \\ p_u - p_v & \text{for } h = r; \end{cases} \tag{2.13}$$

hold.

Proof It is convenient to represent permutation π as $\pi = (\pi_1, u, v, \pi_2)$, where π_1 and π_2 are subsequences of jobs that precede job u and follow job v in permutation π, respectively. Then, $\pi' = (\pi_1, v, u, \pi_2)$.

We present the proof assuming that both sequences π_1 and π_2 are non-empty; otherwise, the corresponding part of the proof can be skipped.

The completion times of all jobs in sequence π_1 are not affected by the swap of jobs u and v, i.e., the equality (2.13) holds for each h, $1 \le h \le r-1$.

Define X as the completion time of the job in the $(r-1)$th position in sequence π (or, equivalently, in π'), i.e., $X = C_{\pi(r-1)} = C_{\pi'(r-1)}$. For $h = r$, we see that $C_{\pi(r)} = C_u(\pi) = X + p_u$ and $C_{\pi'(r)} = C_v(\pi') = X + p_v$, so that due to (2.12), the equality (2.13) holds for $h = r$.

For $h = r+1$, we derive that

$$C_{\pi(r+1)} = C_v(\pi) = X + p_u + p_v;$$
$$C_{\pi'(r+1)} = C_u(\pi') = X + p_v + p_u,$$

so that (2.13) holds for $h = r+1$.

The jobs that follow the position $r+1$ form the same sequence π_2 in both permutations π and π'. Each of these jobs starts in permutation π' exactly at the same time as in permutation π, and therefore, (2.13) holds for each h, $r+2 \le h \le n$.

This proves the theorem. □

For single machine problems to minimize a regular objective function $\Phi(\pi)$ that depends only on the completion times, Theorem 2.3 demonstrates that in a sequence that minimizes $\Phi(\pi)$, the jobs may be arranged in such a way that a job with a larger processing time is not followed by a job with a smaller processing time. Examples of such a function include, but not limited to $\sum C_j^z$ and $\xi C_{\max} + \eta \sum C_j^z$. This immediately implies the following statement.

Theorem 2.4 *For problem* $1||\Phi$, *where* $\Phi \in \left\{\sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, *an optimal permutation can be found by the SPT rule.*

Reformulating Theorem 2.4, we conclude that problem $1||\Phi$, where $\Phi \in \left\{\sum C_j^z, \xi C_{\max} + \eta \sum C_j^z\right\}$, admits the 1-priority $\omega(j) = 1/p_j$.

In Part 2 of this book, we present statements similar to Theorems 2.3 and 2.4 for extended models with changing processing times.

We now formulate the following recipes for proving and disproving that a certain function $\Phi(\pi)$ admits or does not admit 1-priorities.

Recipe 2.1. How to prove that a function $\Phi(\pi)$ admits 1-priorities.

To do this, apply the pairwise interchange technique. Take an arbitrary permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, select a position r, and introduce a permutation π' obtained from π by swapping the elements $\pi(r)$ and $\pi(r+1)$. Derive a function $\omega : N \to \mathbb{R}$ such that for the inequality $\Phi(\pi) \leq \Phi(\pi')$ to hold, it is sufficient that $\omega(\pi(r)) \geq \omega(\pi(r+1))$.

Recipe 2.2. How to disprove that a function $\Phi(\pi)$ does not admit 1-priorities.

If $\Phi(\pi)$ admitted a 1-priority $\omega(j)$, then for any pair of jobs u and v such that $\omega(u) \geq w(v)$, the value of Φ for any permutation π in which u precedes v should be no larger than the value of Φ for permutation π' obtained from π by swapping the jobs u and v. Thus, we should exhibit an instance of the problem of minimizing $\Phi(\pi)$ such that there exist two jobs u and v for which the following holds:

(i) There exists a permutation $(\pi_1, u, \pi_2, v, \pi_3)$ in which job u is scheduled before job v such that $\Phi(\pi_1, u, \pi_2, v, \pi_3) \leq \Phi(\pi_1, v, \pi_2, u, \pi_3)$.

(ii) There exists another permutation $(\hat{\pi}_1, u, \hat{\pi}_2, v, \hat{\pi}_3)$ in which job u is scheduled before job v such that $\Phi(\hat{\pi}_1, u, \hat{\pi}_2, v, \hat{\pi}_3) > \Phi(\hat{\pi}_1, v, \hat{\pi}_2, u, \hat{\pi}_3)$.

2.1.2 Minimizing the Sum of Products

Given two arrays $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, where $a_j \geq 0$ and either $b_j > 1$ or $b_j < 1$ for $1 \leq j \leq n$, and a permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, introduce the expression

$$K(\pi) = \sum_{j=1}^{n} a_{\pi(j)} \prod_{i=j+1}^{n} b_{\pi(i)} \tag{2.14}$$

and consider the problem of minimizing $K(\pi)$ over the set of all permutations, i.e., we are looking for permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ such that the inequality

$$K(\varphi) \leq K(\pi) \tag{2.15}$$

holds for all permutations π.

Associate each j, $1 \leq j \leq n$, with the ratio

$$\kappa(j) = \frac{a_j}{b_j - 1}. \tag{2.16}$$

Theorem 2.5 *Permutation* $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ *such that*

$$\kappa_{\varphi(1)} \leq \kappa_{\varphi(2)} \leq \cdots \leq \kappa_{\varphi(n)} \tag{2.17}$$

satisfies (2.15), i.e., minimizes the expression $K(\pi)$*, provided that all* a_j*'s are non negative and all differences* $b_j - 1$ *are of the same sign.*

Proof The pairwise interchange argument is applied. Suppose that $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ is a permutation that satisfies (2.15), but does not follow the rule (2.17). Then, there exists an index $k < n$ such that

$$\kappa_{\varphi(1)} \leq \cdots \leq \kappa_{\varphi(k-1)} \leq \kappa_{\varphi(k)};\ \kappa_{\varphi(k)} > \kappa_{\varphi(k+1)}.$$

Consider the permutation $\varphi' = (\varphi'(1), \varphi'(2), \ldots, \varphi'(n))$, obtained from φ by swapping the elements $\varphi(k)$ and $\varphi(k+1)$, i.e., φ and φ' satisfy (2.8).

Define $\Delta := K(\varphi) - K\left(\varphi'\right)$. Due to optimality of φ, we must have $\Delta \leq 0$. However,

$$\begin{aligned}
\Delta = &\left(\sum_{j=1}^{k-1} a_{\varphi(j)} \prod_{i=j+1}^{n} b_{\varphi(i)} + a_{\varphi(k)} \prod_{i=k+1}^{n} b_{\varphi(i)} \right. \\
&\left. + a_{\varphi(k+1)} \prod_{i=k+2}^{n} b_{\varphi(i)} + \sum_{j=k+1}^{n} a_{\varphi(j)} \prod_{i=j+1}^{n} b_{\varphi(i)} \right) \\
&- \left(\sum_{j=1}^{k-1} a_{\varphi(j)} \prod_{i=j+1}^{n} b_{\varphi(i)} + a_{\varphi(k+1)} \left(b_{\varphi(k)} \prod_{i=k+2}^{n} b_{\varphi(i)} \right) \right. \\
&\left. + a_{\varphi(k)} \prod_{i=k+2}^{n} b_{\varphi(i)} + \sum_{j=k+1}^{n} a_{\varphi(j)} \prod_{i=j+1}^{n} b_{\varphi(i)} \right)
\end{aligned}$$

$$= a_{\varphi(k)}\big(b_{\varphi(k+1)} - 1\big) \prod_{i=k+2}^{n} b_{\varphi(i)} - a_{\varphi(k+1)}\big(b_{\varphi(k)} - 1\big) \prod_{i=k+2}^{n} b_{\varphi(i)}$$

$$= \prod_{i=k+2}^{n} b_{\varphi(i)}\big(a_{\varphi(k)}\big(b_{\varphi(k+1)} - 1\big) - a_{\varphi(k+1)}\big(b_{\varphi(k)} - 1\big)\big).$$

By the choice of k, we have that

$$\frac{a_{\varphi(k)}}{b_{\varphi(k)} - 1} > \frac{a_{\varphi(k+1)}}{b_{\varphi(k+1)} - 1},$$

Since $\big(b_{\varphi(k)} - 1\big)\big(b_{\varphi(k+1)} - 1\big) > 0$, it follows that $a_{\varphi(k)}\big(b_{\varphi(k+1)} - 1\big) > a_{\varphi(k+1)}\big(b_{\varphi(k)} - 1\big)$, and we deduce that $\Delta > 0$.

Thus, permutation φ cannot be a solution that minimizes $K(\pi)$. Repeating this argument as many times as required, we conclude that for an optimal permutation φ, the condition (2.17) must hold. □

Theorem 2.5 implies that the problem of *minimizing* the sum of products $K(\pi)$ admits a 1-priority $\omega(j) = -\kappa(j) = -a_j/\big(b_j - 1\big)$. If we need to find a permutation that *maximizes* the value of $K(\pi)$, then the corresponding permutation can be found by sorting the values $\kappa(j)$ in non-increasing order.

2.2 Minimizing Total Weighted Completion Time

In this section, we consider problem $1||\sum w_jC_j$, which differs from problem $1||\sum C_j$ discussed in Sect. 2.1.1 by the fact that each job $j \in N$ is given a positive weight w_j. The weights reflect the relative importance of the jobs, so that the jobs with higher weights should be completed earlier.

Suppose that the jobs are sequenced in accordance with some permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. The completion time of job $\pi(j)$ satisfies (2.1), so that the objective function for problem $1||\sum w_jC_j$ can be written as

$$Z(\pi) = \sum_{j=1}^{n} w_{\pi(j)}C_{\pi(j)} = \sum_{j=1}^{n} w_{\pi(j)} \sum_{i=1}^{j} p_{\pi(i)}. \tag{2.18}$$

To solve problem $1||\sum w_jC_j$, we need to find a permutation $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ of jobs such that $Z(\varphi) \le Z(\pi)$ holds for all permutations π. We show that using the pairwise interchange argument, problem $1||\sum w_jC_j$ can be solved by applying the following extension of the SPT priority rule.

Definition 2.3 The jobs of set $N = \{1, 2, \ldots, n\}$ are said to follow the *WSPT* rule (*weighted shortest processing time*) if the jobs are renumbered so that

$$\frac{p_1}{w_1} \leq \frac{p_2}{w_2} \leq \cdots \leq \frac{p_n}{w_n}. \tag{2.19}$$

The following statement holds.

Theorem 2.6 *For problem* $1||\sum w_j C_j$, *an optimal permutation can be found in* $O(n \log n)$ *time, by sequencing the jobs in accordance with the WSPT rule.*

Proof The proof is along the same lines as the proof of Theorem 2.1. Suppose that $\varphi = (\varphi(1), \varphi(2), \ldots, \varphi(n))$ is an optimal permutation of jobs for problem $1||\sum w_j C_j$, but it does not follow the rule (2.19). Then, there exist an index $k < n$ such that

$$\frac{p_{\varphi(1)}}{w_{\varphi(1)}} \leq \cdots \leq \frac{p_{\varphi(k-1)}}{w_{\varphi(k-1)}} \leq \frac{p_{\varphi(k)}}{w_{\varphi(k)}}; \ \frac{p_{\varphi(k)}}{w_{\varphi(k)}} > \frac{p_{\varphi(k+1)}}{w_{\varphi(k+1)}}. \tag{2.20}$$

Consider the permutation $\varphi' = (\varphi'(1), \varphi'(2), \ldots, \varphi'(n))$, obtained from φ by swapping the elements $\varphi(k)$ and $\varphi(k+1)$, i.e., defined by (2.8).

For function $Z(\pi)$ of the form (2.18), define $\Delta := Z(\varphi) - Z(\varphi')$. Due to optimality of φ, we must have $\Delta \leq 0$. However,

$$\begin{aligned}\Delta = &\left(\sum_{j=1}^{k} w_{\varphi(j)} \sum_{i=1}^{j} p_{\varphi(i)} + w_{\varphi(k)} \sum_{i=1}^{k} p_{\varphi(i)} + w_{\varphi(k+1)} \sum_{i=1}^{k+1} p_{\varphi(i)} + \sum_{j=k+2}^{n} w_{\varphi(j)} \sum_{i=1}^{j} p_{\varphi(i)}\right) \\ &\left(-\sum_{j=1}^{k} w_{\varphi(j)} \sum_{i=1}^{j} p_{\varphi(i)} + w_{\varphi(k+1)}\left(\sum_{i=1}^{k-1} p_{\varphi(i)} + p_{\varphi(k+1)}\right)\right. \\ &\left. + w_{\varphi(k)} \sum_{i=1}^{k+1} p_{\varphi(i)} + \sum_{j=k+2}^{n} w_{\varphi(j)} \sum_{i=1}^{j} p_{\varphi(i)}\right) = w_{\varphi(k+1)} p_{\varphi(k)} - w_{\varphi(k)} p_{\varphi(k+1)} > 0,\end{aligned}$$

where the last strict inequality is due to (2.20).

Thus, permutation φ cannot deliver an optimal solution to problem $1||\sum w_j C_j$. Repeating this argument as many times as required, we conclude that for an optimal permutation φ, the condition (2.19) must hold. □

2.3 Minimizing Total Completion Time on Parallel Machines

In this section, we consider the problem of minimizing the sum of the completion times on parallel machines. We show how the results of the previous sections of this chapter can be extended to solve the problems in the classical settings, as well as

their capacitated version in which a machine cannot process more than a predefined number of jobs.

2.3.1 Uniform Machines

We start with the classical problem, traditionally denoted by $Qm||\sum C_j$. Here, each job j of set $N = \{1, 2, \ldots, n\}$ has to be assigned to be processed on one of the $m \geq 2$ parallel uniform machines. Machine M_i has speed s_i. Without loss of generality, we may assume that the machines are numbered in non-increasing order of their speeds, i.e.,

$$s_1 \geq s_2 \geq \ldots \geq s_m. \tag{2.21}$$

It is also convenient to assume that the speed of the slowest machine M_m is equal to 1, and the processing time of job $j \in N$ on machine M_m is equal to p_j. In general, if job j is assigned to be processed on machine M_i, then such processing takes p_j/s_i time units, $1 \leq i \leq m$. A feasible schedule S is determined by

- a partition job N into m subsets $N_1, N_2, \ldots, N_m$, so that the jobs of set N_i and only those are assigned to be processed on machine M_i, $1 \leq i \leq m$;
- the sequence of jobs $\pi^{[i]} = \left(\pi^{[i]}(1), \pi^{[i]}(2), \ldots, \pi^{[i]}(h_i)\right)$ on machine M_i, where $h_i = |N_i|$ and $1 \leq i \leq m$.

Take a machine M_i, $1 \leq i \leq m$, and suppose that in some schedule S, the jobs $\pi^{[i]}(1), \ldots, \pi^{[i]}(h_i)$ are processed on M_i in this order. We have that

$$\begin{aligned} C_{\pi^{[i]}(1)} &= p_{\pi^{[i]}(1)}/s_i, \\ C_{\pi^{[i]}(2)} &= p_{\pi^{[i]}(1)}/s_i + p_{\pi^{[i]}(2)}/s_i, \\ &\ldots \\ C_{\pi^{[i]}(h_i)} &= p_{\pi^{[i]}(1)}/s_i + \ldots + p_{\pi^{[i]}(h_i)}/s_i. \end{aligned}$$

This implies that the contribution of the jobs of set N_i toward the objective function is equal to

$$\sum_{j=1}^{h_i} C_{\pi^{[i]}(j)} = \sum_{j=1}^{h_i} \frac{h_i - j + 1}{s_i} p_{\pi^{[i]}(j)},$$

i.e., an individual contribution for each job assigned to machine M_i is equal to its processing time multiplied by a positional factor of the form k/s_i, where k is a position of the job from the rear of the processing sequence on machine M_i. In order to minimize the total completion time, we need to match the processing times to the n smallest positional factors of the form k/s_i, where $1 \leq k \leq n$ and $1 \leq i \leq m$. As follows from Sect. 2.1, in an optimal schedule, the larger values of the processing times should be matched to smaller positional factors. This can be done by scanning

the jobs in the LPT order and to match the next job to the smallest available positional factor.

Formally, an algorithm for solving problem $Qm||\sum C_j$ can be stated as follows.

Algorithm QSum

INPUT: The processing times $p_1, p_2, \ldots, p_n$ numbered in accordance with the LPT rule (2.11) and the machine speeds $s_1, s_2, \ldots, s_m$ numbered in accordance with (2.21)

OUTPUT: For each machine, M_i, $1 \le i \le m$, permutation $\pi^{[i]}$ that defines the processing sequence of job on the machine in an optimal schedule

Step 1. For each machine M_i, $i = 1, 2, \ldots, m$, define the positional factors $z_i = 1/s_i$ and m empty sequences $\pi^{[i]}$.

Step 2. Scanning the jobs in the order of their numbering, for each job j from 1 to n, do

(a) Find the machine M_v, $1 \le v \le m$, associated with the smallest positional factor, i.e.,

$$z_v := \min\{z_i | 1 \le i \le m\}; \tag{2.22}$$

If such a machine is not unique, break ties by setting v to be the largest index v for which (2.22) holds.

(b) Assign job j to machine M_v and place it in front of the current permutation $\pi^{[v]}$, i.e., define

$$\pi^{[v]} := (j, \pi^{[v]}), \; z_v := z_v + \frac{1}{s_v}.$$

The running time of the algorithm is $O(n \log n)$, including the renumbering of the jobs.

For illustration, consider the following example.

Example 2.2 To test a new scheduling algorithm, a researcher wants to run six sets of test data on three computers. Compared to Computer 3, Computer 1 is three times faster, while Computer 2 is two times faster. The computation time (in minutes) needed to run the test sets on Computer 3 is given in Table 2.2. It is required to organize the computational experiment in such a way that the average completion time of a data set is minimized. Interpreting the three computers as three machines M_1, M_2, and M_3, and the data sets as six jobs, we can reduce the original problem to problem

Table 2.2 Processing times of the tests sets on Computer 3 for Example 2.2

Set j	1	2	3	4	5	6
Processing time p_j	12	36	24	42	18	30

$Q3|\,|\sum C_j$ and solve it by Algorithm QSum. We may assume that the speed of machine M_3 (i.e., that of Computer 3) is taken as 1, so that $s_1 = 3, s_2 = 2$, and $s_3 = 1$. Table 2.3 shows how Algorithm QSum is run. The jobs are scanned in the LPT order. Each row of Table 2.3 shows the parameters after the corresponding job has been assigned to the machine. Notice that when assigning jobs 3 and 5, the current smallest positional factor is not unique, and we break ties giving preference to the machine with the slowest speed (M_3 for job 3 and M_2 for job 5). The last column shows the actual processing times of the jobs, i.e., the initial values p_j from Table 2.2 divided by the speed of the machine the job is assigned to. Figure 2.2 presents a Gantt chart of an optimal schedule. The completion times of the jobs (test sets) are shown in Table 2.4. The average completion time is equal to $(4 + 27 + 24 + 28 + 9 + 14)/6 = 17.667$ min, i.e., 17 min 40 sec.

Notice that Algorithm QSum can be deduced from a solution procedure for a more general scheduling problem $Rm|\,|\sum C_j$ on unrelated parallel machines (see Sect. 4.1.2).

Table 2.3 Running Algorithm QSum for Example 2.2

j	$\pi^{[1]}$	$\pi^{[2]}$	$\pi^{[3]}$	z_1	z_2	z_3	Actual processing times
	()	()	()	$\frac{1}{3}$	$\frac{1}{2}$	1	
4	(4)	()	()	$\frac{2}{3}$	$\frac{1}{2}$	1	14
2	(4)	(2)	()	$\frac{2}{3}$	1	1	18
6	(6, 4)	(2)	()	1	1	1	10
3	(6, 4)	(2)	(3)	1	1	2	24
5	(6, 4)	(5, 2)	(3)	1	$\frac{3}{2}$	2	9
1	(1, 6, 4)	(5, 2)	(3)	$\frac{4}{3}$	$\frac{3}{2}$	2	4

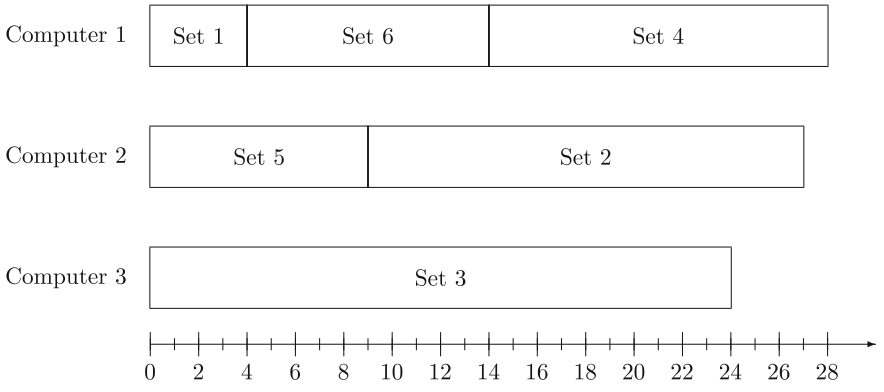

Fig. 2.2 An optimal schedule for Example 2.2

Table 2.4 Completion times of the test sets in an optimal schedule for Example 2.2

Set j	1	2	3	4	5	6
Completion time p_j	4	27	24	28	9	14

Algorithm QSum above can be easily modified to handle a capacitated version of the problem, in which additional restrictions regarding the number of jobs to be assigned to a machine are imposed. Formally, suppose that in any feasible schedule, machine M_i processes no more than $q^{[i]}$ jobs, where $\sum_{i=1}^{m} q^{[i]} \geq n$, so that a feasible schedule exists. We denote the problem under consideration by $Qm\left|\sum_{i=1}^{m} q^{[i]} \geq n\right| \sum C_j$. The algorithm below is a minor modification of Algorithm QSum: If no more jobs can be assigned to a machine, the machine is excluded from consideration by setting the corresponding positional factor to infinity.

Algorithm QSumCap

INPUT: The processing times $p_1, p_2, \ldots, p_n$ numbered in accordance with the LPT rule (2.11) and the machine speeds $s_1, s_2, \ldots, s_m$ numbered in accordance with (2.21) and machine capacities $q^{[1]}, q^{[2]}, \ldots, q^{[m]}$

OUTPUT: For each machine, M_i, $1 \leq i \leq m$, permutation $\pi^{[i]}$ that defines the processing sequence of job on the machine in an optimal schedule

Step 1. For each machine M_i, $i = 1, 2, \ldots, m$, define the positional factors $z_i := 1/s_i$ and m empty sequences $\pi^{[i]}$.

Step 2. Scanning the jobs in the order of their numbering, for each job j from 1 to n, do the following:

(a) Find the machine M_v, $1 \leq v \leq m$, associated with the smallest positional factor that satisfies (2.22). If such a machine is not unique, break ties by setting v to be the largest index v for which (2.22) holds.

(b) Assign job j to machine M_v and place it in front of the current permutation $\pi^{[v]}$, i.e., define $\pi^{[v]} := (j, \pi^{[v]})$. If $z_v = \frac{q^{[v]}}{s_v}$, then exclude this machine from consideration by defining $z_v := +\infty$; otherwise, define $z_v := z_v + \frac{1}{s_v}$.

The running time of Algorithm QSumCap is $O(n \log n)$.

2.3.2 *Identical Machines*

The results of Sect. 2.3.1 can be adapted to solving the problem of minimizing total completion time on identical parallel machines. The speeds of all machines are assumed equal to 1, so that the processing time of job $j \in N$ on any machine is equal to p_j. Problem $Pm|\beta| \sum C_j$ is a special case of problem $Qm|\beta| \sum C_j$, where the

field β either is empty or reads $\sum_{i=1}^{m} q^{[i]} \geq n$. Thus, Algorithms QSum and QSumCap solve the uncapacitated and capacitated versions of the problem on identical machines, respectively.

Given a schedule S, let $F(S) = \sum C_j(S)$ denote the total completion time. For problem $Pm|\,|\sum C_j$ on m parallel machines, a schedule found by Algorithm QSum is denoted by $S^*(m)$. We exclude from consideration the case that $m \geq n$, since it is optimal to assign exactly one job to each of n arbitrarily chosen machines.

Recall that for a real number x, the *ceiling* $\lceil x \rceil$ is equal to the smallest integer that is no less than x. Assuming that the jobs are numbered in accordance with the LPT rule (2.11), in schedule $S^*(m)$, each of the first m jobs takes the last position on one of the machines, each of the next m jobs takes the second from last position on one of the machines, etc. This implies that job j contributes its processing time to the objective function exactly $\left\lceil \frac{j}{m} \right\rceil$ times, so that

$$F(S^*(m)) = \sum_{j=1}^{n} p_j \left\lceil \frac{j}{m} \right\rceil. \tag{2.23}$$

For problem $Pm|\,|\sum C_j$, a solution algorithm can be designed based on different principles, compared to Algorithm QSum. Indeed, an optimal solution can be obtained by scanning the jobs in the SPT order (2.10) and building a schedule from front to rear. Such an algorithm is presented below.

Algorithm PSumSPT

INPUT: The processing times $p_1, p_2, \ldots, p_n$ numbered in accordance with the SPT rule (2.10)

OUTPUT: A schedule S_{SPT} that is defined by permutations $\pi^{[i]}$, $1 \leq i \leq m$

Step 1 For each machine M_i, $i = 1, 2, \ldots, m$, define sequence $\pi^{[i]} := (i)$ that consists of one job i.

Step 2 Scanning the jobs in the order of their numbering, for each job j from $i + 1$ to n, do the following:

For the current partial schedule, by checking the machines in the order of their numbering, determine the machine M_v, $1 \leq v \leq m$, that completes its jobs earlier than other machines. Assign job j to machine M_v, update $\pi^{[v]} := \left(\pi^{[v]}, j\right)$.

Algorithm PSumSPT requires linear time, provided that the SPT sequence of jobs is available.

2.4 Bibliographic Notes

Theorem 2.1 is a classical result that traces back to Hardy et al. (1934), where it is formulated as Theorem 368 on p. 262. Several different proofs are provided for the theorem, starting from that based on the pairwise interchange argument. Another

proof of the theorem is given in Sect. 4.1.3. More proofs of Theorem 2.1 and its extensions can be found in Chap. 6, Section A of the book by Marshall and Olkin (1979).

A link between the problem of minimizing a linear form and a single-flight low-risk helicopter transportation problem is established in Qian et al. (2012). The latter problem arises in the oil and gas offshore mining, when employees are to be delivered to and picked up from a number of offshore installations (rigs or platforms) by a helicopter, so as to minimize the number of people exposed to landings and takeoffs. If the number of people to be delivered to and picked up from an installation j is equal to P_j and D_j, respectively, then an optimal order of visits to the installations can be found by Algorithm Match, so that the installations are visited in non-decreasing order of $P_j - D_j$.

Theorem 2.5 is independently proved by Rau (1971) and Kelly (1982).

Historically, Theorem 2.6 on a priority rule for solving problem $1||\sum w_j C_j$ is one of the first scheduling results. It is proved by Smith (1956), and this is why the WSPT rule stated in Definition 2.3 is often referred to as *Smith's* rule. Queyranne (1993) presents a minimal linear description of the solution polyhedron defined as the convex hull of feasible completion time vectors and deduces Smith's rule using the greedy algorithm known in optimization over polymatroids.

A special case of Theorem 2.4 for problem $1||\sum C_j^2$ is proved in Townsend (1978).

A solution algorithm for problem $Qm||\sum C_j$ is first described in the book by Conway et al. (1967). Our exposition mainly follows the book by Brucker (2007). Algorithm QSumCap for problem $Qm\left|\sum_{i=1}^{m} q^{[i]} \geq n\right|\sum C_j$ is presented in Rustogi and Strusevich (2012).

Algorithm PSumSPT is also first presented in Conway et al. (1967). It should be seen as a version of the famous list scheduling algorithm by Graham (1966) with a list found by the SPT rule.

For an environment with m parallel identical machines with the objective function $\Phi \in \left\{C_{\max}, \sum C_j\right\}$, Rustogi and Strusevich (2013) consider the problem of determining machine impact which shows what can be gained if extra machines are added.

The link between problem $Pm||\sum C_j$ and the low-risk multiflight helicopter transportation problem is established by Qian et al. (2015). That paper presents a number of approximation algorithms for the capacitated version of the problem with pickup only.

While problem $1||\sum w_j C_j$ is polynomially solvable, adding an additional machine changes the complexity dramatically. Indeed, problem $P2||\sum w_j C_j$ is NP-hard even if $p_j = w_j$ for all $j \in N$ (see Bruno et al. (1974)).

References

Brucker P (2007) Scheduling algorithms, 5th edn. Guildford, Springer

Bruno JL, Coffman EG Jr, Sethi R (1974) Scheduling independent tasks to reduce mean finishing time. Comm ACM 17:382–387

Conway RW, Maxwell WL, Miller LW (1967) Theory of scheduling. Addison Wesley, Reading (MA)

Graham RL (1966) Bounds for certain multiprocessing anomalies. Bell Syst Technol J 45:1563–1581

Hardy G, Littlewood JE, Polya G (1934) Inequalities. Cambridge University Press, London

Kelly FP (1982) A remark on search and sequencing problems. Math Oper Res 7:154–157

Marshall AW, Olkin I (1979) Inequalities: Theory of majorization and its applications. Academic Press, New York

Qian F, Gribkovskaia IV, Halskau Ø (2012) Passenger and pilot risk minimization in offshore helicopter transportation. Omega 40:584–593

Qian F, Strusevich VA, Gribkovskaia IV, Halskau Ø (2015) Minimization of passenger takeoff and landing risk in offshore helicopter transportation: models, approaches and analysis. Omega 51:93–106

Queyranne M (1993) Structure of a simple scheduling polyhedron. Math Progr 58:263–285

Rau JG (1971) Minimizing a function of permutations of n integers. Oper Res 19:237–240

Rustogi K, Strusevich VA (2012) Simple matching vs linear assignment in scheduling models with positional effects: a critical review. Eur J Oper Res 222:393–407

Rustogi K, Strusevich VA (2013) Parallel machine scheduling: Impact of adding an extra machine. Oper Res 61:1243–1257

Smith WE (1956) Various optimizers for single state production. Naval Res Logist Quart 3:66–69

Townsend W (1978) The single machine problem with quadratic penalty function of completion times: a barnch-and-bound solution. Manag Sci 24:530–533

http://www.springer.com/978-3-319-39572-2

Scheduling with Time-Changing Effects and Rate-Modifying Activities

Strusevich, V.; Rustogi, K.

2017, XXV, 455 p. 14 illus., Hardcover
ISBN: 978-3-319-39572-2