A combination of flow shop scheduling and the shortest path problem

— **Source link** ☐↗

Kameng Nip, Zhenbo Wang, Fabrice Talla Nobibon, Roel Leus

Institutions: Tsinghua University, Research Foundation - Flanders, Katholieke Universiteit Leuven

Topics: Flow shop scheduling, Job shop scheduling, Shortest path problem, Canadian traveller problem and K shortest path routing

Related papers:

- Combination of Two-Machine Flow Shop Scheduling and Shortest Path Problems

- A study on several combination problems of classic shop scheduling and shortest path

- Combination of parallel machine scheduling and vertex cover

- Efficient approximation algorithms for the routing open shop problem

- Approximation algorithms for the parallel flow shop problem

A combination of flow shop scheduling and the shortest path problem

Nip K, Wang Z, Talla Nobibon F, Leus R.

A Combination of Flow Shop Scheduling and the Shortest Path Problem

Kameng Nip · Zhenbo Wang · Fabrice Talla Nobibon · Roel Leus

Abstract This paper studies a combinatorial optimization problem which is obtained by combining the flow shop scheduling problem and the shortest path problem. The objective of the obtained problem is to select a subset of jobs that constitutes a feasible solution to the shortest path problem, and to execute the selected jobs on the flow shop machines to minimize the makespan. We argue that this problem is NP-hard even if the number of machines is two, and is NP-hard in the strong sense for the general case. We propose an intuitive approximation algorithm for the case where the number of machines is an input, and an improved approximation algorithm for fixed number of machines.

Keywords. approximation algorithm; combination of optimization problems; flow shop scheduling; shortest path.

Kameng Nip
Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, China
E-mail: njm11@mails.tsinghua.edu.cn

Zhenbo Wang (✉)
Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, China
E-mail: zwang@math.tsinghua.edu.cn

Fabrice Talla Nobibon
Postdoctoral researcher for Research Foundation–Flanders
ORSTAT, Faculty of Economics and Business, KU Leuven, Belgium
E-mail: Fabrice.TallaNobibon@kuleuven.be

Roel Leus
ORSTAT, Faculty of Economics and Business, KU Leuven, Belgium
E-mail: Roel.Leus@kuleuven.be

1 Introduction

Combinatorial optimization is an active field in operations research and theoretical computer science. Historically, independent lines separately developed, such as machine scheduling, bin packing, travelling salesman problem, network flows, etc. With the rapid development of science and technology, manufacturing, service and management are often integrated, and decision-makers have to deal with systems involving several characteristics from more than one well-known combinatorial optimization problem. To the best of our knowledge, the combination of optimization problems has received only little attention in literature.

Bodlaender et al. (1994) studied parallel machine scheduling with incompatible jobs, in which two incompatible jobs cannot be processed by the same machine, and the objective is to minimize the makespan. This problem can be considered as a combination of parallel machine scheduling and the coloring problem. Wang and Cui (2012) studied a combination of parallel machine scheduling and the vertex cover problem. The goal is to select a subset of jobs that forms a vertex cover of a given graph and to execute these jobs on m identical parallel machines to minimize the makespan. They proposed an $(3 - \frac{2}{m+1})$-approximation algorithm for this problem. Wang et al. (2013) have investigated a generalization of the above problem that combines the uniformly related parallel machine scheduling problem and a generalized covering problem. They proposed several approximation algorithms and mentioned as future research other combinations of well-known combinatorial optimization problems. This is the core motivation for this work.

Let us consider the following scenario. We aim at building a railway between two specific cities. The railway needs to cross several adjacent cities, which is determined by a map (a graph). The processing time of manufacturing the rail track for each pair of cites varies between the pairs. Manufacturing a rail track between two cities in the graph is associated with a job. The decision-maker needs to make two main decisions: (1) choosing a path to connect the two cities, and (2) deciding the schedule of manufacturing the rail tracks on this path in the factory. In addition, the manufacturing of rail tracks follows several working stages, each stage must start after the completion of the preceding stages, and we assume that there is only one machine for each stage. We wish to accomplish the manufacturing as early as possible, i.e. minimize the last completion time; this is a standard flow shop scheduling problem. How should a decision-maker choose a feasible path such that the corresponding jobs can be manufactured as early as possible? This problem combines the structure of flow shop scheduling and the shortest path problem. Following the framework introduced by Wang et al. (2013), we can regard our problem as a combination of those two problems.

Finding a simple path between two vertices in a directed graph is a basic problem that can be polynomially solved (Ahuja et al., 1993). Furthermore, if we want to find a path under a certain objective, various optimization problems come within our range of vision. The most famous one is the classic

shortest path problem, which can be solved in polynomial time if the graph contains no negative cycle, and otherwise it is NP-hard (Ahuja et al., 1993). Moreover, many optimization problems have a similar structure. For instance, the min-max shortest path problem (Kouvelis and Yu, 1997) studies a problem with multiple weights associated with each arc, and the objective is to find a directed path between two specific vertices such that the value of the maximum among all its total weights is minimized. The multi-objective shortest path problem (Warburton, 1987) also has multiple weights, but the objective is to find a Pareto optimal path between two specific vertices to satisfy some specific objective function.

Flow shop scheduling is one of the three basic models of multi-stage scheduling (the others are open shop scheduling and job shop scheduling). Flow shop scheduling with the objective of minimizing the makespan is usually denoted by $Fm||C_{max}$, where m is the number of machines. In one of the earliest papers on scheduling problems, Johnson (1954) showed that $F2||C_{max}$ can be solved in $O(n \log n)$ time, where n is the number of jobs. On the other hand, Garey et al. (1976) proved that $Fm||C_{max}$ is strongly NP-hard for $m \geq 3$.

The contributions of this paper include: (1) a formal description of the considered problem, (2) the argument that the considered problem is NP-hard even if $m = 2$, and NP-hard in the strong sense if $m \geq 3$, and (3) several approximation algorithms.

The rest of the paper is organized as follows. In Section 2, we first give a formal definition of the problem stated above, then we briefly review flow shop scheduling and some shortest path problems, and introduce some related algorithms that will be used in the subsequent sections. In Section 3, we study the computational complexity of the combined problem. Section 4 provides several approximation algorithms for this problem. Some concluding remarks are provided in Section 5.

2 Preliminaries

2.1 Problem Description

We first give a formal definition of our problem, which is a combination of the flow shop scheduling problem and the shortest path problem.

Definition 1 Given a directed graph $G = (V, A)$ with two distinguished vertices $s, t \in V$ and m flow shop machines, each arc $a_j \in A$ corresponds with a job $J_j \in J$ with processing times $(p_{1j}, p_{2j}, \cdots, p_{mj})$ respectively. The $Fm|$shortest path$|C_{max}$ problem is to find a $s - t$ directed path P, and to schedule the jobs of J_P on the flow shop machines to yield the minimum makespan over all P, where J_P denotes the set of jobs corresponding to the arcs in P.

The considered problem is a combination of flow shop scheduling and the classic shortest path problem, mainly because the two optimization problems

are special cases of this problem. For example, consider the following instances with $m = 2$. If there is a unique path from s to t in G, as shown in the left of Fig. 1, our problem is the two-machine flow shop scheduling problem. If all the processing times on the second machine are zero, as shown in the right of Fig. 1, then our problem is equivalent to the classic shortest path with respect to the processing times on the first machine. These examples illustrate that these two optimization problems are inherent in the considered problem.
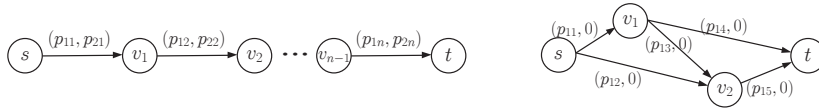

Fig. 1 Special cases of our problem

In this paper, we will use the results of some optimization problems that have a similar structure with the classic shortest path problem. We introduce the following generalized shortest path problem.

Definition 2 Given a directed graph $G = (V, A, w^1, \cdots, w^K)$ and two distinguished vertices $s, t \in V$ with $|A| = n$. Each arc $a_j \in A, j = 1, \cdots, n$ is associated with K weights $w_j^1, \cdots, w_j^K$, and we define the vector $w^k = (w_1^k, w_2^k, \cdots, w_n^k)$ for $k = 1, 2, \cdots, K$. The goal of our shortest path problem $SP(G, s, t, f)$ is to find a $s-t$ directed path P that minimizes $f(w^1, w^2, \cdots, w^K, x)$, in which f is a given objective function and $x \in \{0, 1\}^n$ contains the decision variables such that $x_j = 1$ if and only if $a_j \in P$.

For ease of exposition, we use SP instead of $SP(G, s, t, f)$ when there is no danger of confusion. Notice that SP is a generalization of various shortest path problems. For instance, if we consider $K = 1$ and $f(w^1, x) = w^1 \cdot x$, where $\cdot$ is the dot product, this problem is the classic shortest path problem. If $K = 2$ and $f(w^1, w^2, x) = \min\{w^1 \cdot x : w^2 \cdot x \leq W\}$, where W is a given number, this problem is the shortest weight-constrained path problem (Garey and Johnson, 1979). If $f(w^1, w^2, \cdots, w^K, x) = \max\{w^1 \cdot x, w^2 \cdot x, \cdots, w^K \cdot x\}$, the problem is the min-max shortest path problem (Kouvelis and Yu, 1997). In the following sections, we will analyze our combined problem by setting appropriate weights and objective function in SP.

2.2 Algorithms for Flow Shop Scheduling Problems

First, we introduce some trivial bounds for flow shop scheduling. Denote by C_{max} the makespan of an arbitrary flow shop schedule with job set J. A feasible shop schedule is called dense when any machine is idle if and only if there is no job that can be processed at that time on that machine. For arbitrary dense flow shop schedule, we have

$$C_{max} \geq \max_{i \in \{1, \cdots, m\}} \left\{ \sum_{J_j \in J} p_{ij} \right\}, \tag{1}$$

and

$$C_{max} \leq \sum_{J_j \in J} \sum_{i=1}^{m} p_{ij}. \tag{2}$$

For each job, we have

$$C_{max} \geq \sum_{i=1}^{m} p_{ij}, \qquad \forall J_j \in J. \tag{3}$$

In flow shop scheduling problems, a schedule is called a permutation schedule if all jobs are processed in the same order on each machine. Conway et al. (1971) proved that there always exists a permutation schedule which is optimal for $F2||C_{max}$ and $F3||C_{max}$. In a permutation schedule, the critical job and critical path are important concepts for the analysis of related algorithms.

Suppose we are given a job set J with n jobs. Let $\sigma = (\sigma(1), \cdots, \sigma(n))$ be a permutation of $(1, \cdots, n)$ for a three-machine (or two-machine) flow shop, and let $\{J_{\sigma(1)}, J_{\sigma(2)}, \cdots, J_{\sigma(n)}\}$ be the corresponding schedule. For simplicity of notation, we denote the permutation and the schedule by $(1, 2, \cdots, n)$ and $\{J_1, J_2, \cdots, J_n\}$ respectively. A directed graph is defined as follows. We define a vertex (i, j) with an associated weight $p_{i,j}$ for each job J_j and each machine M_i, for $i = 1, 2, 3$ (or $i = 1, 2$) and $j = 1, 2, \cdots, n$. We include arcs leading from each vertex (i, j) towards $(i + 1, j)$, and from (i, j) towards $(i + 1, j + 1)$ for $j = 1, \cdots, n - 1$. The total weight of a maximum weight path from $(1, 1)$ to $(3, n)$ (or $(2, n)$), which is called a critical path, is equal to the makespan of the corresponding permutation schedule. For the three-machine case, the critical jobs with respect to σ are defined as the jobs J_u and J_v such that $(1, u), (2, u), (2, v)$ and $(3, v)$ appear in the critical path, i.e. the jobs J_u and J_v satisfy

$$C_{max} = \sum_{j=1}^{u} p_{1j} + \sum_{j=u}^{v} p_{2j} + \sum_{j=v}^{n} p_{3j}. \tag{4}$$

For the two-machine case, the critical job with respect to σ is defined as the job J_ν such that $(1, \nu)$ and $(2, \nu)$ appear in the critical path, i.e. the job J_ν satisfies

$$C_{max} = \sum_{j=1}^{\nu} p_{1j} + \sum_{j=\nu}^{n} p_{2j}. \tag{5}$$

Johnson (1954) proposed a sequencing rule for $F2||C_{max}$, which is one of the oldest results of the scheduling literature, and is commonly referred to as Johnson's rule.

In Johnson's rule, jobs are scheduled as early as possible. This rule produces a permutation schedule, and Johnson showed that this schedule is optimal. Notice that this schedule is obtained in $O(n \log n)$ time.

Algorithm 1 Johnson's rule

1: Set $S_1 = \{J_j \in J | p_{1j} \leq p_{2j}\}$ and $S_2 = \{J_j \in J | p_{1j} > p_{2j}\}$.
2: Process the jobs in S_1 first in a non-decreasing order of p_{1j}, and then schedule the jobs in S_2 in a non-increasing order of p_{2j}; ties may be broken arbitrarily.

For the general problem $Fm||C_{max}$, Gonzalez and Sahni (1978) first presented an $\lceil \frac{m}{2} \rceil$-approximation algorithm that runs in $O(mn \log n)$ time by solving $\lceil \frac{m}{2} \rceil$ two-machine flow shop scheduling problems. Röck and Schmidt (1982) proposed an alternative approach by reducing the original problem to an artificial two-machine flow shop problem; this approach is called machine aggregation heuristic. They obtained a permutation by solving the artificial problem in $O(mn + n \log n)$ time, and proved that it has the same performance guarantee of $\lceil \frac{m}{2} \rceil$. Based on the machine aggregation heuristic, Chen et al. (1996) proposed an algorithm for $F3||C_{max}$ with an improved performance guarantee of $\frac{5}{3}$. In the same paper, they also modified the Gonzalez and Sahni's algorithm if m is odd, by partitioning the machines into $\frac{m-3}{2}$ two-machine flow shop scheduling problems, and one three-machine flow shop scheduling problem which was solved by their $\frac{5}{3}$-approximation algorithm. The modified algorithm has the same performance ratio $\frac{m}{2}$ if m is even, and an improved ratio $\frac{m}{2} + \frac{1}{6}$ if m is odd. It is known that a PTAS exists for $Fm||C_{max}$ (Hall, 1988).

We refer to the aggregation heuristic of Röck and Schmidt (1982) as the RS algorithm, and we will use it later to derive an algorithm for our combined problem. The RS algorithm can be described as follows for the three-machine case.

Algorithm 2 The RS algorithm for $F3||C_{max}$

1: Construct an artificial two-machine flow shop scheduling problem with processing times $a_j = p_{1j} + p_{2j}$ on the first machine and $b_j = p_{2j} + p_{3j}$ on the second machine for $J_j \in J$. Implement Johnson's rule to obtain an optimal permutation σ for the two-machine problem.
2: Assign the jobs on the three machines according to σ as early as possible. Denote the makespan of this permutation schedule as C_{max}.
3: **return** σ, C_{max}.

The running time of this algorithm is $O(n \log n)$, which is the same as Johnson's rule. Notice that the algorithm returns a permutation schedule, and hence the resulting makespan C_{max} satisfies the equality (4).

2.3 Algorithms for Shortest Path Problems

In this paper, we will use the following two results of the shortest path problems. The first one is the well-known Dijkstra's algorithm, which solves the classic shortest path problem with nonnegative edge weights in $O(|V|^2)$ time

(Dijkstra, 1959). The second one is a FPTAS result for the min-max shortest path problem, which is presented by Aissi, Bazgan and Vanderpooten (2006). Kouvelis and Yu (1997) first proposed min-max criteria for several problems, including the shortest path problem. Aissi, Bazgan and Vanderpooten (2006) studied the computational complexity and proposed several approximation schemes. The min-max shortest path problem with K weights $w_j^1, \cdots, w_j^K$ associated with each arc a_j, is to find a path P between two specific vertices that minimizes $\max_{k \in \{1, \cdots, K\}} \sum_{a_j \in P} w_j^k$. It was shown that this problem is NP-hard even for $K = 2$, and that a FPTAS exists if K is a fixed number (Warburton, 1987; Aissi, Bazgan and Vanderpooten, 2006). The algorithm of Aissi, Bazgan and Vanderpooten (2006), referred to the ABV algorithm in this paper, is based on dynamic programming and scaling techniques. The following result implies that the ABV algorithm is a FPTAS if K is a constant.

Theorem 1 (Aissi, Bazgan and Vanderpooten (2006)) *Given an arbitrary positive value $\epsilon > 0$, in a given directed graph with K nonnegative weights associated with each arc, a directed path P between two specific vertices can be found by the ABV algorithm with the property*

$$\max_{i \in \{1, 2, \cdots, K\}} \left\{ \sum_{a_j \in P} w_j^i \right\} \leq (1 + \epsilon) \max_{i \in \{1, 2, \cdots, K\}} \left\{ \sum_{a_j \in P'} w_j^i \right\}$$

for any other path P' between the two specific vertices, and the running time is $O(|A||V|^{K+1}/\epsilon^K)$.

3 Computational Complexity of $Fm|\text{shortest path}|C_{max}$

In this section, we study the computational complexity of our problem. First, it is straightforward that $Fm|\text{shortest path}|C_{max}$ is NP-hard in the strong sense if $m \geq 3$, as a consequence of the fact that $Fm||C_{max}$ is a special case of our problem.

On the other hand, although $F2||C_{max}$ and the classical shortest path are polynomially solvable, we argue that $F2|\text{shortest path}|C_{max}$ is NP-hard. We prove this result by using a reduction from the NP-complete problem PARTITION (Garey and Johnson, 1979). Our proof is similar to the well-known NP-hardness proof for the shortest weight-constrained path problem (Batagelj et al., 2000).

Theorem 2 *$Fm|\text{shortest path}|C_{max}$ is NP-hard even if $m = 2$, and is NP-hard in the strong sense for $m \geq 3$.*

Proof We only need to prove the first part. It is easy to see that the decision version of $F2|\text{shortest path}|C_{max}$ belongs to NP. Consider an arbitrary instance of PARTITION with $S = \{a_1, \cdots, a_n\}$ with size $s(a_k) \in \mathbb{Z}^+$ for each k, and let $C = \sum_{a \in S} s(a)/2$. We now construct the directed graph $G = (V, A, W)$
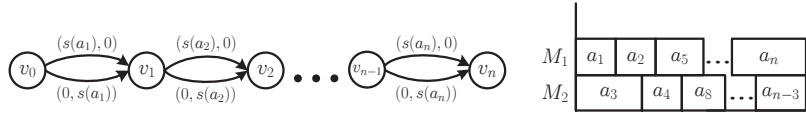
Fig. 2 The reduction from PARTITION to $F2|$shortest path$|C_{max}$

and the corresponding jobs. The graph has $n + 1$ vertices $v_0, v_1, \cdots, v_n$, each pair of $(v_k, v_{k+1}), k = 0, 1, \cdots, n - 1$, is joined by two parallel arcs (jobs) with processing times $(s(a_{k+1}), 0)$ and $(0, s(a_{k+1}))$ respectively, both leading from vertex v_k towards v_{k+1} (see the left of Fig. 2). We wish to find the jobs corresponding to a path from v_0 to v_{n+1}. It is not difficult to check that there is a feasible schedule with makespan not more than C if and only if there is a partition of set S (see the right of Fig. 2). Therefore, the decision version of $F2|$shortest path$|C_{max}$ is NP-complete. $\square$

4 Approximation Algorithms

4.1 An intuitive Algorithm

To start off, we propose an intuitive algorithm for $Fm|$shortest path$|C_{max}$. The main idea of this algorithm is to set $K = 1$ and $f = w^1 \cdot x$ in SP, i.e. to find a classical shortest path with one specific set of weights. An intuitive setting is $w_j^1 = \sum_{i=1}^m p_{ij}$ for each arc. We find the shortest path with respect to w^1 by Dijkstra's algorithm, and then schedule the corresponding jobs on the flow shop machines. We refer to this algorithm as the FD algorithm. The subsequent analysis will show that the performance ratio of the FD algorithm remains the same for an arbitrarily selected flow shop scheduling algorithm that provides a dense schedule, regardless of the performance ratio of the algorithm.

Algorithm 3 The FD algorithm

1: Find the shortest path in G with weights $w_j^1 := \sum_{i=1}^m p_{ij}$ by Dijkstra's algorithm. For the returned path P, construct the job set J_P.
2: Obtain a dense schedule of the jobs of J_P by an arbitrary flow shop scheduling algorithm. Let σ be the returned job schedule and C_{max} the returned makespan, and denote the job set J_P by S.
3: **return** S, σ **and** C_{max}

It is straightforward that the total running time of the FD algorithm is $O(|V|^2 + T(m, n))$, where $T(m, n)$ is the running time of the flow shop scheduling algorithm. Therefore, suppose the flow shop scheduling algorithm we used is polynomial time, then the FD algorithm is polynomial time even if m is an

input of the instance. Before we analyze the performance of this algorithm, we first introduce some notations. Let J^* be the set of jobs in an optimal solution, and C_{max}^* be the corresponding makespan, and let S and C_{max} be those returned by the FD algorithm respcetively.

Theorem 3 *The FD algorithm is m-approximate, and this bound is tight.*

Proof By the lower bound (1) introduced in Section 2.2, we have

$$mC_{max}^* \geq \sum_{J_j \in J^*} \sum_{i=1}^{m} p_{ij}. \tag{6}$$

Since the returned path is a shortest path with respect to w^1, by (2) we have

$$C_{max} \leq \sum_{J_j \in S} \sum_{i=1}^{m} p_{ij} = \sum_{J_j \in S} w_j^1 \leq \sum_{J_j \in J^*} w_j^1 = \sum_{J_j \in J^*} \sum_{i=1}^{m} p_{ij}. \tag{7}$$

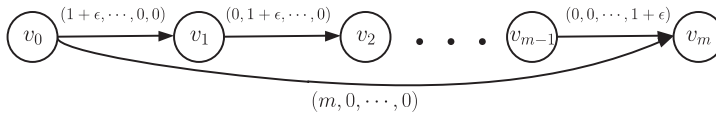By combining (6) with (7), it follows that $C_{max} \leq mC_{max}^*$.

Fig. 3 Tight example for the FD algorithm

Consider the instance shown in Fig. 3. We wish to find a path from vertex v_0 to v_m. The makespan returned by the FD algorithm is $C_{max} = m$ with the arc (v_0, v_m), whereas the makespan of an optimal schedule is $C_{max}^* = 1 + \epsilon$ with the other arcs. Notice that there is only one job in the returned solution, hence the returned makespan remains m regardless of the algorithm used for the flow shop scheduling. The bound is tight because $\frac{C_{max}}{C_{max}^*} \to m$ when $\epsilon \to 0$. $\square$

4.2 An Improved Algorithm for Fixed m

In this subsection, we assume that m, the number of flow shop machines, is a constant. Instead of finding an optimal shortest path from s to t with respect to specific weights, we implement the ABV algorithm mentioned in Section 2.3, which will return a $(1 + \epsilon)$-approximated solution for the min-max shortest path problem. In other words, we will set $K = m$ and use the objective function $f = \max\{w^1 \cdot x, w^2 \cdot x, \cdots, w^K \cdot x\}$ in SP, where the wights $w^1, w^2, \cdots, w^K$ will be decided later.

Inspired by the work of Gonzalez and Sahni (1978) and Chen et al. (1996), we proceed as follows: after obtaining a feasible path by the ABV algorithm, we schedule the corresponding jobs by partitioning the m machines into several groups. Denote the machine as M_i, $i = 1, \cdots, m$ (indexed following the routing of the flow shop). More specifically, we partition the m machines into m_3 groups of three consecutive machines in the routing M_{3i-2}, M_{3i-1}, M_{3i} ($i = 1, \cdots, m_3$), m_2 groups of two consecutive machines in the routing M_{3m_3+2i-1}, M_{3m_3+2i} ($i = 1, \cdots, m_2$), and m_1 individual machines $M_{3m_3+2m_2+i}$ ($i = 1, \cdots, m_1$), in which the value of m_1, m_2, m_3 will be derived later. For the three-machine subproblems on M_{3i-2}, M_{3i-1} and M_{3i} ($i = 1, \cdots, m_3$), we implement the RS algorithm to obtain the permutations. For the two-machine subproblems on M_{3m_3+2i-1} and M_{3m_3+2i} ($i = 1, \cdots, m_2$), we implement Johnson's rule to obtain the permutations. The permutations for the single-machine subproblems are arbitrary. Then we form a schedule for the original m-machine problem, in which the sequences of jobs on machines M_i are the permutations obtained above, and are executed as early as possible. Notice the property that an optimal schedule is always a permutation schedule only stands for $F2||C_{max}$ and $F3||C_{max}$, and the performance guarantee relies on the properties of critical jobs as we will see in the subsequent analysis. The reason why we partition the m machines in this particular fashion is related to this fact, as will be explained below.

The main idea of our algorithm is described as follows. We initially set the weights $(w_j^1, w_j^2, \cdots, w_j^m) = (p_{1j}, p_{2j}, \cdots, p_{mj})$. The algorithm iteratively runs the ABV algorithm and the above partition scheduling algorithm (the values of m_1, m_2, m_3 will be decided later) by adopting the following revision policy: in a current schedule, if there exists a job whose weight is large enough with respect to the current makespan, we will modify the weights of arcs corresponding to large jobs to $(M, M, \cdots, M)$, where M is a sufficient large number, and then mark these jobs. The algorithm terminates if no such job exists. Another termination condition is that a marked job appears in a current schedule. We return the schedule with minimum makespan among all current schedules as the solution of the algorithm. We refer to this algorithm as the PAR algorithm. Notice that the weights of arcs may vary in each iteration, whereas the processing times of jobs remain the same throughout this algorithm.

Before we formally state the PAR algorithm, we first provide more details about the parameter choices. For $m = 2$ and $m = 3$, by following the subsequent analysis of the performance of this algorithm, one can verify that the best possible performance ratio is $\frac{3}{2}$ and 2 respectively. An intuitive argument is that the best possible performance ratio for the general case of the PAR algorithm is $\rho = m_1 + \frac{3}{2}m_2 + 2m_3$. For a given m, as m_1, m_2, m_3 are nonnegative integers, our task is to minimize $m_1 + \frac{3}{2}m_2 + 2m_3$ such that $m_1 + 2m_2 + 3m_3 = m$. A simple calculation yields the following result:

$$(m_1, m_2, m_3) = \begin{cases} (0, 0, \frac{m}{3}) & \text{if } m = 0 \pmod 3, \\ (1, 0, \frac{m-1}{3}) & \text{if } m = 1 \pmod 3, \\ (0, 1, \frac{m-2}{3}) & \text{if } m = 2 \pmod 3, \end{cases} \qquad (8)$$

and

$$\rho = \begin{cases} \frac{2m}{3} & \text{if } m = 0 \pmod 3, \\ \frac{2m+1}{3} & \text{if } m = 1 \pmod 3, \\ \frac{4m+1}{6} & \text{if } m = 2 \pmod 3. \end{cases} \qquad (9)$$

In other words, the best way is to partition the machines in such a way that we have a maximum number of three-machine subsets. The pseudocode of the PAR algorithm is described by Algorithm 4.

Algorithm 4 The PAR algorithm

1: Derive (m_1, m_2, m_3) and ρ using (8) and (9).
2: Initially, $(w_j^1, w_j^2, \cdots, w_j^m) := (p_{1j}, p_{2j}, \cdots, p_{mj})$ for each arc $a_j \in A$ corresponding to $J_j \in J$.
3: Given $\epsilon > 0$, implement the ABV algorithm to obtain a path P for SP, and construct the corresponding job set as J_P.
4: Partition the m machines: m_3 three-machines subsets $(M_{3i-2}, M_{3i-1}, M_{3i}, \ i = 1, \cdots, m_3)$; one two-machine subsets $(M_{m-1}$ and $M_m)$ if $m_2 = 1$; one single-machine subset (M_m) if $m_1 = 1$.
5: Run RS algorithm to obtain the permutations for these three-machine flow shops, and Johnson's rule to obtain the permutation for the two-machine flow shop. Let the sequence of the single-machine problem be arbitrary.
6: For the original problem, schedule the jobs of J_P according to those permutations on each machine as early as possible. Denote the returned makespan as C'_{max}, and the job schedule as σ'.
7: $S := J_P$, $\sigma := \sigma'$, $C_{max} := C'_{max}$, $D := \emptyset$, $M := (1 + \epsilon) \sum_{J_j \in J} \sum_{i=1}^m p_{ij} + 1$.
8: **while** $J_P \cap D = \emptyset$ **and** there exists a job J_j in J_P such that $\sum_{i=1}^m p_{ij} > \frac{C'_{max}}{\rho}$ **do**
9:     **for** all jobs with $\sum_{i=1}^m p_{ij} > \frac{C'_{max}}{\rho}$ in $J \backslash D$ **do**
10:       $(w_j^1, w_j^2, \cdots, w_j^m) := (M, M, \cdots, M)$, $D := D \cup \{J_j\}$.
11:     **end for**
12:     Implement the ABV algorithm to obtain a path P to SP, and construct the corresponding job set as J_P.
13:     Schedule the jobs of J_P by the rule described in lines $4 - 6$.
14:     **if** $C'_{max} < C_{max}$ **then**
15:       $S := J_P$, $\sigma := \sigma'$, $C_{max} := C'_{max}$.
16:     **end if**
17: **end while**
18: **return** S, σ **and** C_{max}.

It is easy to see that the PAR algorithm will return a feasible solution of $Fm|$shortest path$|C_{max}$. We now discuss its computational complexity. Let the total number of jobs be $|A| = n$. Notice that the weights of arcs can be revised at most n times. It is straightforward that the total running time of the PAR algorithm is $O(n^2|V|^{m+1}/\epsilon^m + mn^2 \log n)$, since there are at most n

iterations, in which the running time of the ABV algorithm is $O(n|V|^{m+1}/\epsilon^m)$ and scheduling takes $O(mn \log n)$ time. If m and ϵ are fixed numbers, then the PAR algorithm is a polynomial time algorithm.

Let J^* be the set of jobs in an optimal solution, and C^*_{max} the corresponding makespan, and let S and C_{max} be those returned by the PAR algorithm respectively. The following theorem shows the performance of the PAR algorithm.

Theorem 4 *Given $\epsilon > 0$, the worst-case ratio of the PAR algorithm for $Fm|$shortest path$|C_{max}$ is*

$$(1+\epsilon)\rho = \begin{cases} (1+\epsilon)\frac{2m}{3} & \text{if } m = 0 \pmod 3, \\ (1+\epsilon)\frac{2m+1}{3} & \text{if } m = 1 \pmod 3, \\ (1+\epsilon)\frac{4m+1}{6} & \text{if } m = 2 \pmod 3. \end{cases} \tag{10}$$

Proof We will distinguish two different cases: $J^* \cap D \neq \emptyset$ and $J^* \cap D = \emptyset$.

Case 1 $J^* \cap D \neq \emptyset$

In this case, there is at least one job in the optimal solution, say J_j, such that $C'_{max} < \rho \sum_{i=1}^m p_{ij}$ holds for a current schedule with makespan C'_{max} during the execution. Notice that the schedule returned by the PAR algorithm is the schedule with minimum makespan among all current schedules, and we have $C_{max} \leq C'_{max}$. It follows from (3) that

$$C_{max} \leq C'_{max} < \rho \sum_{i=1}^m p_{ij} \leq \rho C^*_{max}. \tag{11}$$

Case 2 $J^* \cap D = \emptyset$

Consider the last current schedule during the execution of the algorithm. Denote the corresponding job set and the makespan as J' and C'_{max} respectively.

In this case, we first argue that $J' \cap D = \emptyset$. Suppose that this is not the case, i.e. $J' \cap D \neq \emptyset$. Since $J^* \cap D = \emptyset$, we know the weights of arcs corresponding to the jobs in J^* have not been revised. Hence we have $(1+\epsilon) \max_{i \in \{1, \cdots, m\}} \left\{ \sum_{J_j \in J^*} w_j^i \right\} < M$. Moreover, by the assumption $J' \cap D \neq \emptyset$, we have $\max_{i \in \{1, \cdots, m\}} \left\{ \sum_{J_j \in J'} w_j^i \right\} \geq M$. By Theorem 1, the solution returned by the ABV algorithm satisfies

$$M \leq \max_{i \in \{1, \cdots, m\}} \left\{ \sum_{J_j \in J'} w_j^i \right\} \leq (1+\epsilon) \max_{i \in \{1, \cdots, m\}} \left\{ \sum_{J_j \in J^*} w_j^i \right\} < M,$$

which leads to a contradiction.

Remember that in the PAR algorithm, the machines are divided into three parts, namely three-machines subsets together with at most one two-machine subset or a single machine. We solve these subproblems by the RS algorithm, Johnson's rule and an arbitrary algorithm respectively. It is clear that the sum

of the makespans of those schedules is an upper bound for C'_{max}. Denote C^2_{max} and J^2_ν as the makespan and the critical job of the two-machine subproblem returned by Johnson's rule, and let the corresponding machines be M_{i_2}, M_{i_2+1}. Denote C^3_{max} and J^3_u, J^3_v as the makespan and the critical jobs returned by the RS algorithm for the three-machine subproblems with largest makespan, and let the machines be $M_{i_3}, M_{i_3+1}, M_{i_3+2}$. Denote the single machine as M_{i_1}, on which the total processing time is $\sum_{J_j \in J'} p_{i_1 j}$.

For the two-machine case, suppose that $p_{i_2,\nu} \geq p_{i_2+1,\nu}$. Noticing that $p_{i_2,j} \geq p_{i_2+1,j}$ for the job scheduled after J_ν in the schedule returned by Johnson's rule and form (5), it follows that

$$C^2_{max} \leq \sum_{J_j \in J'} p_{i_2,j} + p_{i_2+1,\nu} \leq \sum_{J_j \in J'} p_{i_2,j} + \frac{1}{2}(p_{i_2,\nu} + p_{i_2+1,\nu}). \qquad (12)$$

For the three-machine case, we study two subcases corresponding with $u = v$ and $u < v$ for the critical jobs.

Subcase 2.1 $u = v$.

Consider the schedule with respect to C^3_{max}. We can rewrite (4) as

$$C^3_{max} = \sum_{j=1}^{u-1} p_{i_3,j} + p_{i_3,u} + p_{i_3+1,u} + p_{i_3+2,u} + \sum_{j=u+1}^{n} p_{i_3+2,j}. \qquad (13)$$

Suppose that the processing times of the critical jobs of the three-machine subproblem satisfy $p_{i_3,u} \geq p_{i_3+2,u}$, thus we have $p_{i_3 u} + p_{i_3+1,u} \geq p_{i_3+1,u} + p_{i_3+2,u}$, i.e. $a_u \geq b_u$ for the artificial two-machine flow shop in the RS algorithm. Since the RS algorithm schedules the jobs by Johnson' rule, thus we have $a_j \geq b_j$ for the jobs scheduled after J_u, i.e. $p_{i_3,j} \geq p_{i_3+2,j}$. From (13), we have

$$C^3_{max} \leq \sum_{J_j \in J'} p_{i_3,j} + p_{i_3,u} + p_{i_3+1,u} + p_{i_3+2,u}. \qquad (14)$$

Since $J' \cap D = \emptyset$, we know the weights of arcs corresponding to the jobs in the last current schedule have not been revised, and $\sum_{i=1}^{m} p_{ij} \leq \frac{C'_{max}}{\rho}$ for each job $J_j \in J'$, since otherwise the algorithm will continue. Since $J^* \cap D = \emptyset$, the weights of arcs corresponding to the jobs in this optimal schedule have not been revised. Thus, it follows from (1), (3), Theorem 1, (12), (14) and the fact that the schedule returned by the PAR algorithm is the schedule with

minimum makespan among all current schedules, that

$$C_{max} \le C'_{max} \le m_3 C^3_{max} + m_2 C^2_{max} + m_1 \sum_{J_j \in J'} p_{i_1,j}$$

$$\le m_3 \left(\sum_{J_j \in J'} p_{i_3,j} + p_{i_3,u} + p_{i_3+1,u} + p_{i_3+2,u} \right)$$

$$+ m_2 \left(\sum_{J_j \in J'} p_{i_2,j} + \frac{1}{2}(p_{i_2,\nu} + p_{i_2+1,\nu}) \right) + m_1 \sum_{J_j \in J'} p_{i_1,j}$$

$$\le m_3 \left((1+\epsilon) \max_{i \in \{1,\cdots,m\}} \left\{ \sum_{J_j \in J^*} p_{ij} \right\} + \frac{C'_{max}}{\rho} \right)$$

$$+ m_2 \left((1+\epsilon) \max_{i \in \{1,\cdots,m\}} \left\{ \sum_{J_j \in J^*} p_{ij} \right\} + \frac{C'_{max}}{2\rho} C'_{max} \right)$$

$$+ m_1 (1+\epsilon) \max_{i \in \{1,\cdots,m\}} \left\{ \sum_{J_j \in J^*} p_{ij} \right\}$$

$$\le (m_1 + m_2 + m_3)(1+\epsilon) C^*_{max} + \left(\frac{m_2}{2\rho} + \frac{m_3}{\rho} \right) C'_{max}$$

Substituting (8) and (9) into the m_1, m_2, m_3 and ρ, by a simple calculation, we arrive at

$$C_{max} \le C'_{max} \le (1+\epsilon)\rho C^*_{max}. \tag{15}$$

Subcase 2.2 $u < v$.

We also assume that $p_{i_3,u} \ge p_{i_3+2,u}$, and $p_{i_2,\nu} \ge p_{i_2+1,\nu}$, an argument similar to the previous case shows that the jobs scheduled after J_v satisfies $p_{i_3,j} \ge p_{i_3+2,j}$. Since $u < v$, it follows from (4) that

$$C^3_{max} \le \sum_{J_j \in J'} p_{i_3,j} + \sum_{j=u}^{v} p_{i_3+1,j} \le \sum_{J_j \in J'} p_{i_3,j} + \sum_{J_j \in J'} p_{i_3+1,j}. \tag{16}$$

Similarly, it is not difficult to show that

$$C_{max} \le C'_{max} \le m_3 C^3_{max} + m_2 C^2_{max} + m_1 \sum_{J_j \in J'} p_{i_1,j}$$

$$\le m_3 \left(\sum_{J_j \in J'} p_{i_3,j} + \sum_{J_j \in J'} p_{i_3+1,j} \right)$$

$$+ m_2 \left(\sum_{J_j \in J'} p_{i_2,j} + \frac{1}{2}(p_{i_2,\nu} + p_{i_2+1,\nu}) \right) + m_1 \sum_{J_j \in J'} p_{i_1,j}$$

$$\le (m_1 + m_2 + 2m_3)(1+\epsilon) C^*_{max} + \frac{m_2}{2\rho} C'_{max}$$

Substituting (8) and (9) into m_1, m_2, m_3 and ρ, by a simple calculation, we obtain

$$C_{max} \leq C'_{max} \leq (1+\epsilon)\rho C^*_{max}. \qquad (17)$$

For the cases where the last current schedule has critical jobs satisfying $p_{i_2,\nu} < p_{i_2+1,\nu}$ or $p_{i_3,u} < p_{i_3+2,u}$, analogous arguments would yield the same result.

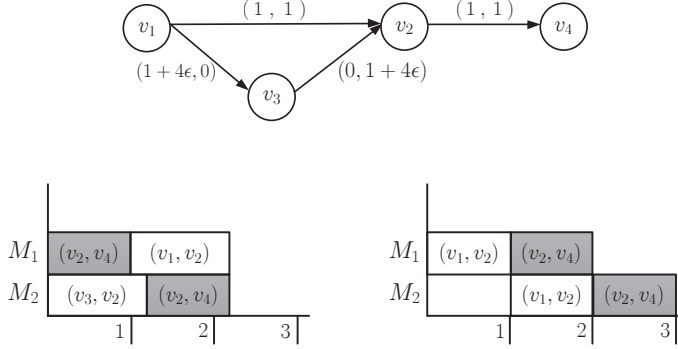Now we show that the performance ratio of the PAR algorithm cannot less than ρ. First, we propose two instances for $m = 2$ and $m = 3$.

Fig. 4 Example for $m = 2$

If $m = 2$, the performance ratio of the PAR algorithm is $\frac{3}{2}(1+\epsilon)$. Consider the following instance shown in Fig. 4. We wish to find a path from v_1 to v_4. Notice that the ABV algorithm returns the path with arcs (v_1, v_2) and (v_2, v_4), and the corresponding makespan C'_{max} by Johnson's rule is 3. All the corresponding jobs satisfy $p_{1j} + p_{2j} = 2 \leq \frac{2}{3}C'_{max}$, and thus the algorithm terminates. Therefore, the makespan of the returned schedule by the PAR algorithm is $C_{max} = 3$ (see the right schedule of Fig. 4). On the other hand, the optimal makespan is $C^*_{max} = 2 + 4\epsilon$ with arcs (v_1, v_3), (v_3, v_2) and (v_2, v_4) (see the left schedule of Fig. 4). The worst case ratio of the PAR algorithm cannot be less than $\frac{3}{2}$ as $\frac{C_{max}}{C^*_{max}} \to 3/2$ when $\epsilon \to 0$ for this instance.

For the case where $m = 3$, the performance ratio of the PAR algorithm is $2(1+\epsilon)$. Consider the instance shown in Fig. 5. We wish to find a path from vertex v_1 to v_6. Notice that the ABV algorithm returns the path with arcs $(v_1, v_4) \to (v_4, v_5) \to (v_5, v_6)$. The makespan of the schedule returned by the RS algorithm is $C'_{max} = 4$. All the corresponding jobs satisfy $p_{1j} + p_{2j} = 2 \leq \frac{1}{2}C'_{max}$, and thus the algorithm terminates. Therefore, the makespan of the schedule returned by the PAR algorithm is $C_{max} = 4$ (see the right schedule of Fig. 5). On the other hand, the makespan of an optimal job schedule is $C^*_{max} = 2(1+\epsilon)^2$, by selecting the arcs (v_1, v_2), (v_2, v_3), and (v_3, v_6) (see the
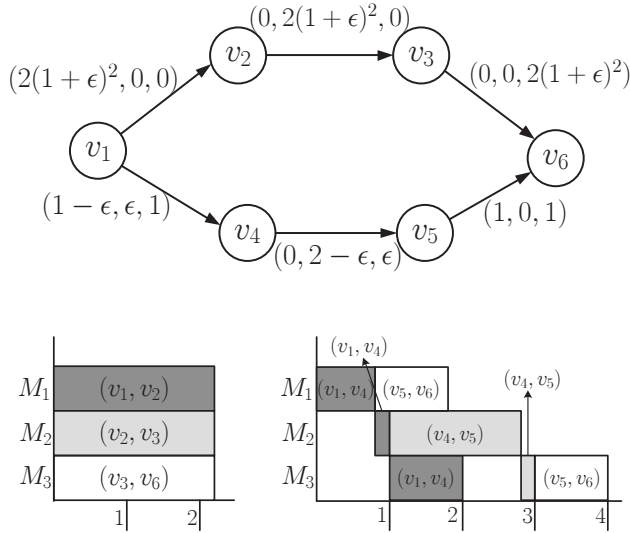
Fig. 5 Example for $m = 3$

left schedule of Fig. 5). The worst case ratio of the PAR algorithm cannot be less than 2 as $\frac{C_{max}}{C^*_{max}} \to 2$ when $\epsilon \to 0$ for this instance.

By extending and modifying the above examples to the general case, the instance described in Fig. 6 can be used to show that the performance ratio of the PAR algorithm cannot be less than ρ. □

5 Conclusions

This paper has studied a combination problem of flow shop scheduling and the shortest path problem. We show the hardness of this problem, and present some approximation algorithms. For future research, it would be interesting to find an approximation algorithm with a better performance ratio for this problem. The question whether $F2|$shortest path$|C_{max}$ is NP-hard in the strong sense is still open. One can also consider the combination of other combinatorial optimization problems. All these questions deserve further investigation.
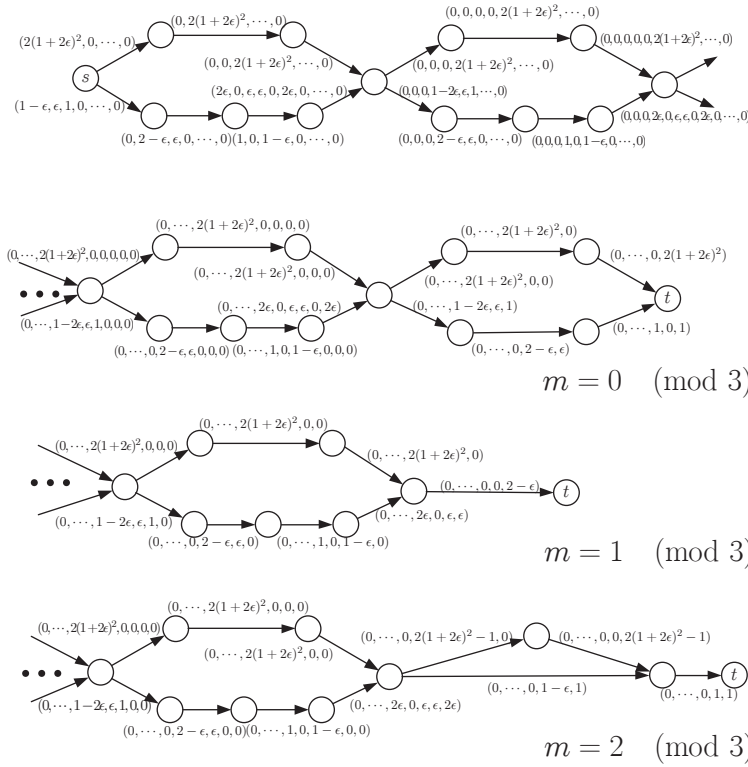
Acknowledgments

Fig. 6 Example for fixed m

References

Ahuja RK, Magnanti TL, Orlin JB (1993) Network Flows: Theory, Algorithms, and Applications. Prentice Hall, New Jersey

Aissi H, Bazgan C, Vanderpooten D (2006) Approximating min-max (regret) versions of some polynomial problems. In: Chen, D., Pardolos, P.M. (eds.) COCOON 2006, LNCS, vol 4112, pp 428–438. Springer, Heidelberg

Batagelj V, Brandenburg FJ, Mendez P, Sen A (2000) The generalized shortest path problem. CiteSeer Archives

Bodlaender HL, Jansen K, Woeginger GJ (1994) Scheduling with incompatible jobs. Disc Appl Math, 55: 219–232

Chen B, Glass CA, Potts CN, Strusevich VA (1996) A new heuristic for three-machine flow shop scheduling. Oper Res 44: 891–898

Conway RW, Maxwell W, Miller L (1967) Theory of scheduling. Reading

Dijkstra EW (1959) A note on two problems in connexion with graphs. Numer Math 1: 269–271

Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and job-shop scheduling. Math Oper Res 1: 117–129

Garey MR, Johnson DS (1979) Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco

Gonzalez T, Sahni S (1978) Flowshop and jobshop schedules: complexity and approximation. Oper Res 26: 36–52

Hall LA (1998) Approximability of flow shop scheduling. Math Program 82: 175–190

Johnson SM (1954) Optimal two- and three-stage production schedules with setup times included. Nav Res Logist Q 1: 61–68

Kouvelis P, Yu G (1997) Robust discrete optimization and its applications. Kluwer Academic Publishers, Boston

Röck H, Schmidt G (1982) Machine aggregation heuristics in shop scheduling. Method Oper Res 45: 303–314

Wang Z, Cui Z (2012) Combination of parallel machine scheduling and vertex cover. Theor Comput Sci 460: 10–15

Wang Z, Hong W, He D (2013): Combination of parallel machine scheduling and covering problem. Working paper, Tsinghua University

Warburton A (1987) Approximation of pareto optima in multiple-objective, shortest-path problems. Oper Res 35: 70–79