Combinations of Some Shop Scheduling Problems and the Shortest Path Problem: Complexity and Approximation Algorithms

Kameng Nip, Zhenbo Wang* and Wenxun Xing

Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, China

Abstract. We consider several combinatorial optimization problems which combine the classic shop scheduling problems, namely open shop scheduling or job shop scheduling, and the shortest path problem. The objective of the obtained problem is to select a subset of jobs that forms a feasible solution of the shortest path problem, and to execute the selected jobs on the open (or job) shop machines to minimize the makespan. We show that these problems are NP-hard even if the number of machines is two, and cannot be approximated within a factor less than 2 if the number of machines is an input unless P = NP. We present several approximation algorithms for these combination problems.

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

1 Introduction

Combinatorial optimization has been developed for more than fifty years, and it involves many active subfields, e.g. network flows, scheduling, bin packing, etc. Usually these subfields were arisen by different applications or theoretical interests, and separately developed. The advancement of science and technology makes it possible to integrate manufacturing, service and management. At the same time, the decision-makers always need to deal with problems incurred by more than one combinatorial optimization problems.

Wang and Cui [17] introduced a problem combining two classic combinatorial optimization problems, namely parallel machine scheduling and the vertex cover problem. The combination problem is to select a subset of jobs that forms a vertex cover and to schedule it on some identical parallel machines such that the makespan is minimized. They proposed a \((3 - \frac{2}{m+1})\)-approximation algorithm. This work also inspired the study of the combination of different combinatorial optimization problems.

Flow shop, open shop and job shop are three basic models of multi-stage scheduling problems. Nip and Wang [12] studied a combination problem that combines two-machine flow shop scheduling and the shortest path problem.

* Corresponding author. Department of Mathematical Sciences, Tsinghua University, Beijing, 100084, China. Email: zwang@math.tsinghua.edu.cn
They argued that this problem is NP-hard, and proposed two approximation algorithms with worst-case ratio 2 and \( \frac{3}{2} \) respectively. Recently they extended the results to the case that the number of flow shop machines is arbitrary \[13\]. One motivation of this problem is manufacturing rail racks. We need to build a railway between two cities. How should we choose a feasible path in a map, such that the corresponding rail tracks (jobs) can be manufactured on some flow shop machine as early as possible? It is convincing to change the flow shop environment into the other two well-known shop environments, i.e. open shop and job shop, as they also apply widely in the real world. This is the core motivation for this current work. In this paper, we mainly study two types of problems: the combination of open shop scheduling and the shortest path problem, and the combination of job shop scheduling and the shortest path problem.

The contributions of this paper are described as follows: (1) we argue that these combination problems are NP-hard even if the number of machines is two, and if the number of machines is an input, these problems cannot be approximated within a factor of 2 unless P = NP; (2) we present several approximation algorithms with performance ratio summarized as follows in which \( \epsilon > 0 \) is any constant and \( \mu \) is the maximum number of operations per job in job shop scheduling.

| Number of Machines | Open Shop | Job Shop |
|--------------------|-----------|----------|
| 2                  | FPTAS     | \( \frac{3}{2} + \epsilon^* \) |
| \( m \) (fixed)    | PTAS**    | \( O\left(\frac{1}{\log \log (m \mu)}\right) \) |
| \( m \) (input)    | \( m \)   | \( m \)   |

* Assume that each job has at most 2 operations.
** A \((2 + \epsilon)\)-approximation algorithm is also proposed.

The rest of the paper is organized as follows. In Section \[2\] we give a formal definition of the combination problems stated above, and briefly review some related problems and algorithms will be used subsequently. In Section \[3\] we study the computational complexity of these combination problems and give an inapproximability result when the number of machines is an input. Section \[4\] provides several approximation algorithms for these problems. Some concluding remarks are provided in Section \[5\].

## 2 Preliminaries

### 2.1 Problem Description

We first define the combination problems considered in this paper.

**Definition 1.** Given a directed graph \( G = (V, A) \) with two distinguished vertices \( s, t \in V \), and \( m \) machines. Each arc \( a_j \in A \) corresponds with a job \( J_j \in J \).
Each job $J_i$ has several operations $O_{1j}, O_{2j}, \ldots, O_{sj}$ (in the open shop, $s = m$ and the order is arbitrary; in the job shop, the order is given as a chain). The processing times for $J_i$ on machine $M_i$ is $p_{ij}$. The $O_m|\text{shortest path}|C_{\text{max}}$ ($Jm|\text{shortest path}|C_{\text{max}}$) problem is to find a $s-t$ directed path $P$ of $G$, and to schedule the jobs of $J_P$ on the open (job) shop machines to yield the minimum makespan over all $P$, where $J_P$ denotes the set of jobs corresponding to the arcs in $P$.

We denote the number of jobs (arcs) as $n$, i.e. $|A| = |J| = n$. Denote by \{${M_1, M_2, \ldots, M_m}$\} the $m$ machines, and let $\mu_{ij}$ be the times of $J_i$ needed to be processed on $M_i$. Notice $\mu_{ij} = 1$ in the open shop.

It is not difficult to see that the open (job) shop scheduling problem and the classic shortest path problem are special cases of our problems, and hence we say the considered problems are the combinations of the scheduling problems and the shortest path problem. We will show that the combination problems appear different aspects in computational complexity and algorithm design from the shop scheduling problems or the shortest path 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 generalized shortest path problem defined in [12], and extend it to $K$ weights.

**Definition 2.** Given a directed graph $G = (V, A, w^1, \ldots, w^K)$ and two distinguished vertices $s, t \in V$ with $|A| = n$. Each arc $a_j \in A, j = 1, \ldots, n$ is associated with $K$ weights $w^1_j, \ldots, w^K_j$, and we define vector $w^k = (w^k_1, w^k_2, \ldots, w^k_n)$ for $k = 1, 2, \ldots, K$. The goal of our shortest path problem $SP(G, s, t, f)$ is to find a $s-t$ directed path $P$ that minimizes $f(w^1, w^2, \ldots, w^K; x)$, 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 simplicity of notation, we denote $SP$ instead of $SP(G, s, t, f)$ in the rest of the paper. Notice $SP$ is a generalization of various shortest path problems. For example, if we set $K = 1$ and $f(w^1, x) = w^1 \cdot x$, where $\cdot$ is the dot product, it is the classic shortest path problem. If $f(w^1 \cdot x, w^2 \cdot x, \ldots, w^K \cdot x, x) = \max\{w^1 \cdot x, w^2 \cdot x, \ldots, w^K \cdot x\}$, it is the min-max shortest path problem [1].

### 2.2 Review of Open Shop and Job Shop Scheduling

Gonzalez and Sahni [1] first gave a linear time optimal algorithm for $O2|C_{\text{max}}$. They also proved that $O\mu|C_{\text{max}}$ is NP-hard for $m \geq 3$, however whether it is strongly NP-hard is still an outstanding open problem. A feasible shop schedule is called dense when any machine is idle if and only if there is no job that could be processed on it. Rácsmány (see Bárány and Fiala [2]) observed that for any dense schedule, the makespan is at most twice of the optimal solution, that leads to a greedy algorithm. Sebastianov and Woeginger [15] presented a PTAS for fixed $m$, which is obtained by dividing jobs into large jobs and small jobs. Their algorithm first optimally schedules the large jobs, then fills the operations of the
small jobs into the ‘gaps’. In this paper, we will use these algorithms, and refer to the GS algorithm, Ráczmány algorithm and the SW algorithm respectively. We present the main results of these algorithms as follows.

**Theorem 1** ([5]). The GS algorithm returns an optimal schedule for $O_2||C_{\text{max}}$ in linear time such that $C_{\text{max}} = \max \left\{ \max_{j \in J} (p_{1j} + p_{2j}), \sum_{j \in J} p_{1j}, \sum_{j \in J} p_{2j} \right\}$.

**Theorem 2** ([2, 16]). Ráczmány algorithm returns a 2-approximation algorithm for $O_m||C_{\text{max}}$ such that $C_{\text{max}} \leq \sum_{j \in J} p_{lj} + \sum_{i=1}^{m} p_{ik} \leq 2C_{\text{max}}$, where $J_k$ is the last completed job and processed on $M_l$.

**Theorem 3** ([15]). The SW algorithm is a PTAS for $O_m||C_{\text{max}}$.

For job shop schedule with an unlimited number of jobs, few polynomially solvable cases are known. One is $J_2|\text{op} \leq 2|C_{\text{max}}$, which can be solved by Jackson’s rule [3] that is an extension of Johnson’s rule for $F_2||C_{\text{max}}$ [8], where $\text{op} \leq 2$ means there are at most 2 operations per job. The idea is to divide the jobs into two sets according to the processing order of the jobs, and implement Johnson’s rule for each job set, then combine the schedule. In fact, a slightly change may lead to NP-hard problems. For instance, $J_2|\text{op} \leq 3|C_{\text{max}}$ and $J_3|\text{op} \leq 2|C_{\text{max}}$ are NP-hard [9]. $J_2|p_{ij} \in \{1, 2\}|C_{\text{max}}$ and $J_3|p_{ij} = 1|C_{\text{max}}$ are strongly NP-hard [10]. For the general case $J|m||C_{\text{max}}$, Shmoys, Stein and Wein [16] constructed a randomized approximation algorithm with worst-case ratio $O\left(\log^2(m\mu) \log \log(m\mu)\right)$, where $\mu$ is the maximum number of operations per job. Schmidt, Siegel and Srinivasan [14] obtained a deterministic algorithm with the same bound by derandomizing. We refer to it as the SSW-SSS algorithm. Moreover, for fixed $m$, the best known approximation algorithm is also proposed in [16] with an approximation factor $2 + \epsilon$, where $\epsilon > 0$ is an arbitrary constant. If $\mu$ is a constant, the problem is denoted as $J_m|\text{op} \leq \mu|C_{\text{max}}$ that admits a PTAS [7]. We list the main results mentioned above as follows.

**Theorem 4** ([6]). Jackson’s rule solves $J_2|\text{op} \leq 2|C_{\text{max}}$ in $O(n \log n)$ time.

**Theorem 5** ([14, 16]). The SSW-SSS algorithm solves $J_m||C_{\text{max}}$ in polynomial time, and return a schedule with makespan

$$O\left(\log^2(m\mu) \log \log(m\mu) \left( \max_{i \in \{1, \ldots, m\}} \sum_{j \in J} \mu_{ij}p_{ij} + \max_{J_j \in J} \sum_{i=1}^{m} \mu_{ij}p_{ij} \right) \right).$$

Furthermore, a well-known inapproximability result is that $O||C_{\text{max}}, F||C_{\text{max}}$ and $J||C_{\text{max}}$ cannot be approximated within $\frac{5}{4}$ unless $P = NP$ [18]. Recently, Mastrolilli and Svensson [11] showed that $J||C_{\text{max}}$ cannot be approximated within $O(\log(m\mu)^{1-\epsilon})$ for $\epsilon > 0$ based on a stronger assumption than $P \neq \text{NP}$. 
To conclude this subsection, we list some trivial bounds for a dense shop schedule. Denote by $C_{\text{max}}$ the makespan of an arbitrary dense shop schedule with job set $J$, and we have

$$C_{\text{max}} \geq \max_{i \in \{1, \ldots, m\}} \left\{ \sum_{j \in J} \mu_{ij} p_{ij} \right\},$$

(1)

and

$$C_{\text{max}} \leq \sum_{j \in J} \sum_{i=1}^{m} \mu_{ij} p_{ij}.$$

(2)

For each job, we have

$$C_{\text{max}} \geq \sum_{i=1}^{m} \mu_{ij} p_{ij}, \quad \forall J_j \in J.$$

(3)

### 2.3 Review of Shortest Path Problems

It is well-known that Dijkstra’s algorithm solves the classic shortest path problem with nonnegative edge weights in $O(|V|^2)$ time [3]. We have mentioned the min-max shortest path problem, that is NP-hard even for $K = 2$, and Aissi, Bazgan and Vanderpooten proposed a FPTAS if $K$ is a fixed number [1]. We refer to their algorithm as the ABV algorithm, which has the following result.

**Theorem 6** ([1]). Given $\epsilon > 0$, in a directed graph with $K$ nonnegative weights on each arc, where $K$ is a fixed number. The ABV algorithm finds a path $P$ between two specific vertices satisfying

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

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

In this paper, sometimes we need to find the min-mix shortest path among all the paths visiting some specified arcs if such a path exists. We propose a modified ABV algorithm for this problem in Appendix A.

### 3 Computational Complexity

First, notice that $Om||C_{\text{max}}$ and $Jm||C_{\text{max}}$ are special cases of the corresponding combination problems, thus the combination problem is not easier than its component optimization problems. On the other hand, we know $O2||C_{\text{max}}$ and $J2|op \leq 2|C_{\text{max}}$ are polynomially solvable, but we can simply verify that the corresponding combination problems, say $O2|\text{shortest path}|C_{\text{max}}$ and $J2|op \leq 2|\text{shortest path}|C_{\text{max}}$, are NP-hard by adopting the same reduction proposed in [12] for the NP-hardness of $F2|\text{shortest path}|C_{\text{max}}$. We summarize the results as Theorem 7.
Theorem 7. Even if $m = 2$, $Jm|\text{shortest path}|C_{\text{max}}$ is strongly NP-hard and $O \text{m}|\text{shortest path}|C_{\text{max}}$ is NP-hard. $J2|\text{op} \leq 2, \text{shortest path}|C_{\text{max}}$ is NP-hard.

Now we consider the case where the number of machines $m$ is part of the input. Williamson et al. showed that it is NP-hard to approximate $O|\text{max}, F|\text{max}$ or $J|\text{max}$ within a factor less than $\frac{3}{4}$ by a reduction from the restricted versions of 3-SAT \[18\]. They also showed that deciding if there is a scheduling of length at most 3 is in $\mathbf{P}$. We show that for these problems combining with shortest path problem, deciding if there is a scheduling of length at most 1 is still NP-hard. Our proof is established by constructing a reduction from 3-Dimensional Matching (3DM) that is NP-complete \[4\].

3-DIMENSIONAL MATCHING:

**Instance:** Sets $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$, $C = \{c_1, \ldots, c_n\}$, and a family $F = \{T_1, \ldots, T_m\}$ of triples with $|T_k \cap A| = |T_k \cap B| = |T_k \cap C| = 1$ for $k = 1, \ldots, m$. Assume that $m \geq n$ without loss of generality.

**Question:** Does $F$ contains a matching, i.e. a subfamily $F'$ for which $|F'| = n$ and $\cup_{T_k \in F'} T_k = A \cup B \cup C$?

**Theorem 8.** For $O|\text{shortest path}|C_{\text{max}}$, deciding if there is a scheduling of length at most 1 is NP-hard.

**Proof.** Given an instance of 3DM, we construct an instance of $O|\text{shortest path}|C_{\text{max}}$ with $2n + m$ machines. For $i = 1, \ldots, n$, machines $M_i$, $M_{n+i}$ and $M_{2n+i}$ correspond to $a_i$, $b_i$ and $c_i$ respectively, and the remained $m - n$ machines denoted by $M_{3n+1}, M_{3n+2}, \ldots, M_{2n+m}$ are 'dummy' machines. The graph has $3m + 1$ vertices, denoted by $v_{1,a}, v_{1,b}, v_{1,c}, v_{2,a}, v_{2,b}, v_{2,c}, \ldots, v_{m,a}, v_{m,b}, v_{m,c}, v_{m+1,a}$. For $k = 1, \ldots, m$, there are arcs $(v_{k,a}, v_{k,b}), (v_{k,b}, v_{k,c})$ and $(v_{k,c}, v_{k+1,a})$ corresponding to jobs $J^a_k, J^b_k, J^c_k$. Let $p(J^j_k)$, $j = a, b, c$ and $k = 1, \ldots, m$, be a $(2n + m)$-dimensional vector whose $i$-th component corresponds to the processing time of $J^j_k$ on $M_i$. Let $e_i$ be a $(2n + m)$-dimensional vector with 1 for the $i$-th component and 0s for the others. Now we can define the processing times of the jobs:

$p(J^a_k) = e_i$ if $a_i \in T_k$;

$p(J^b_k) = e_{n+i}$ if $b_i \in T_k$;

$p(J^c_k) = e_{2n+i}$ if $c_i \in T_k$.

Selecting jobs $J^a_k, J^b_k, J^c_k$ implies that $T_k$ is in the matching. Moreover for $k = 1, \ldots, m$, there are $m - n$ parallel arcs from $v_{k,a}$ towards $v_{k+1,a}$, corresponding to jobs $J^a_k, J^b_k, \ldots, J^m_k$ with processing times $e_{3n+1}, e_{3n+2}, \ldots, e_{2n+m}$ respectively.

Selecting those jobs implies that $T_k$ is not in the matching. The objective is to find a path from $v_{1,a}$ to $v_{n+1,a}$ and to schedule the corresponding jobs (arcs) such that the makespan is at most 1, that completes the reduction. One example is shown in Figure 1.

It can be verified that a schedule with makespan at most 1 if and only if there is a matching for 3DM, and then the result follows. The details are deferred to the full version.

Notice that the reduction in Theorem 8 is also valid for $F|\text{shortest path}|C_{\text{max}}$ and $J|\text{shortest path}|C_{\text{max}}$, since each job in the reduction has only one nonzero processing time. Therefore we have the following result.
Corollary 1. $O||C_{\text{max}}$, $F||C_{\text{max}}$ and $J||C_{\text{max}}$ do not admit an approximation algorithm with worst-case ratio less than 2, unless P = NP.

To our knowledge, the best known inapproximability results based on P $\neq$ NP for $F||C_{\text{max}}$, $O||C_{\text{max}}$ and $F||C_{\text{max}}$ are still $\frac{5}{4}$. The corollary implies that the combination problems of the three shop scheduling problems and the shortest problem have stronger inapproximability results.

4 Approximation Algorithms

4.1 An Intuitive Algorithms for Arbitrary $m$

An intuitive algorithm was proposed for $F_2|\text{shortest path}|C_{\text{max}}$ in [12]. The idea is to find the classic shortest path by setting the weight of an arc be the sum of processing times of its corresponding job, and then schedule the returned jobs by Johnson’s rule. This simple idea can be extended to all the combination problems we considered, even if the number of machines is an input.

Algorithm 1 The SD algorithm for $O|\text{shortest path}|C_{\text{max}}$ ($J|\text{shortest path}|C_{\text{max}}$)

1: Find the shortest path in $G$ with weights $w_j^1 := \sum_{i=1}^{m} \mu_{ij} p_{ij}$ by Dijkstra algorithm.

For the returned path $P$, construct the job set $J_P$.

2: Obtain a dense schedule for the jobs of $J_P$ by an arbitrary open (job) shop scheduling algorithm. Let $\sigma$ be the returned job schedule and $C_{\text{max}}$ the returned makespan, and denote the job set $J_P$ by $S$.

3: return $S$, $\sigma$ and $C_{\text{max}}$.

It is easy to show that Algorithm 1 is a $m$-approximation algorithm, by the bounds [1], [2], [3] and the fact that the returned path is the shortest path with respect to the single weight of each arc.
4.2 A Unified Algorithms for Fixed $m$

In [12], a $\frac{3}{2}$-approximation algorithm was proposed for $F2|\text{shortest path}|C_{\text{max}}$. The idea is to iteratively find a feasible path by the ABV algorithm with two weights for each arc and schedule the corresponding jobs by Johnson’s rule, and then adaptively modified the weights of arcs. We generalize this idea to solve the combination problems considered in this paper. We first propose a unified framework which denoted as UAR($Alg$, $\rho$, $m$), where $Alg$ is a polynomial time algorithm used for shop scheduling, $\rho$ is a control parameter to decide the termination rule of the iterations and the jobs to be modified, and $m$ is the number of machines.

Algorithm 2 Algorithm UAR($Alg$, $\rho$, $m$)

1: Initially, $(w_1^j, w_2^j, \cdots, w_m^j) := (\mu_1^j p_{1j}, \mu_2^j p_{2j}, \cdots, \mu_m^j p_{mj})$, for $a_j \in A$ corresponding to $J_j$.
2: Given $\epsilon > 0$, implement the ABV algorithm to obtain a feasible path $P$ to $SP$, and construct the corresponding job set as $J_P$.
3: Schedule the jobs of $J_P$ by the algorithm $Alg$, denote the returned makespan as $C'_{\text{max}}$, and the job schedule as $\sigma'$.
4: $S := J_P$, $\sigma := \sigma'$, $C_{\text{max}} := C'_{\text{max}}$, $D := \emptyset$, $M := (1 + \epsilon) \sum_{j \in J} \sum_{i=1}^{m} \mu_{ij} p_{ij} + 1$.
5: while $J_P \cap D = \emptyset$ and there exists $J_j$ in $J_P$ satisfies $\sum_{i=1}^{m} \mu_{ij} p_{ij} \geq \rho C_{\text{max}}$ do
6: for all jobs satisfy $\sum_{i=1}^{m} \mu_{ij} p_{ij} \geq \rho C_{\text{max}}$ in $J \setminus D$ do
7: $(w_1^j, w_2^j, \cdots, w_m^j) := (M, M, \cdots, M)$, $D := D \cup \{J_j\}$.
8: end for
9: Implement the ABV algorithm to obtain a feasible path $P$ to $SP$, and construct the corresponding job set as $J_P$.
10: Schedule the jobs of $J_P$ by the algorithm $Alg$, denote the returned makespan as $C''_{\text{max}}$, and the job schedule as $\sigma''$.
11: if $C''_{\text{max}} < C_{\text{max}}$ then
12: $S := J_P$, $\sigma := \sigma''$, $C_{\text{max}} := C''_{\text{max}}$.
13: end if
14: end while
15: return $S$, $\sigma$ and $C_{\text{max}}$.

By setting the appropriate scheduling algorithms and control parameters, we can derive algorithms for different combination problems. Notice that at most $n$ jobs are modified in the UAR($Alg$, $\rho$, $m$) algorithm, therefore the iterations execute at most $n$ times. Since the scheduling algorithms for shop scheduling and the ABV algorithms are all polynomial time algorithms (for fixed $m$ and $\epsilon$), we claim that the following algorithms based on UAR($Alg$, $\rho$, $m$) are polynomial-time algorithms. We present the algorithms and their performance as follows and the detailed proofs are given in Appendix B.

We first apply the UAR($Alg$, $\rho$, $m$) algorithm to $O2|\text{shortest path}|C_{\text{max}}$ by setting $Alg$ be the GS algorithm and $\rho = 1$, and refer to this algorithm as the GAR algorithm.
Algorithm 3 The GAR algorithm for \(O_2|\text{shortest path}|C_{\text{max}}\)
1: Set \(m = 2\), \(Alg\) be the GS algorithm for \(O_2||C_{\text{max}}\) and \(\rho = 1\).
2: Solve the problem by implementing \(\text{UAR}(Alg, \rho, m)\).

Theorem 9. The GAR algorithm is a FPTAS for \(O_2|\text{shortest path}|C_{\text{max}}\).

For \(Om|\text{shortest path}|C_{\text{max}}\) where \(m\) is fixed, based on \(\text{UAR}(Alg, \rho, m)\) and Ráczmány algorithm, we obtain the following algorithm, referred to the RAR algorithm by considering an appropriate \(\rho\).

Algorithm 4 The RAR algorithm for \(Om|\text{shortest path}|C_{\text{max}}\)
1: Set \(Alg\) be Ráczmány algorithm for \(Om||C_{\text{max}}\) and \(\rho = \frac{1}{2}\).
2: Solve the problem by implementing \(\text{UAR}(Alg, \rho, m)\).

Theorem 10. Given \(\epsilon > 0\), the RAR algorithm is a \((2 + \epsilon)\)-approximation algorithm for \(Om|\text{shortest path}|C_{\text{max}}\).

The framework also can be applied to the combination problem of job shop scheduling and the shortest path problem. For the combination of \(J_2|op \leq 2|C_{\text{max}}\) and the shortest path problem, we obtain a \((\frac{3}{2} + \epsilon)\)-approximation algorithm by implementing Jackson’s rule and setting \(\rho = \frac{2}{3}\) in the \(\text{UAR}(Alg, \rho, m)\) algorithm. We refer to this algorithm as the JJAR algorithm, and describe it in Algorithm 5. Remind that all \(\mu_{ij} = 1\) in \(J_2|op \leq 2|C_{\text{max}}\).

Algorithm 5 The JJAR algorithm for \(J_2|op \leq 2, \text{shortest path}|C_{\text{max}}\)
1: Set \(m = 2\), \(Alg\) be Jackson’s rule for \(J_2|op \leq 2|C_{\text{max}}\) and \(\rho = \frac{2}{3}\).
2: Solve the problem by implementing \(\text{UAR}(Alg, \rho, m)\).

Before studying the worst-case performance of the JJAR algorithm, we establish the following lemma. Let \((1 \to 2)\ ((2 \to 1))\) indicate the order that a job needs to be processed on \(M_1\) (\(M_2\)) first and then on \(M_2\) (\(M_1\)).

Lemma 1. For \(J_2|op \leq 2|C_{\text{max}}\), let \(C^J_{\text{max}}\) be the makespan returned by Jackson’s rule. Suppose we change the processing order of all jobs to be \((1 \to 2)\ ((2 \to 1))\), and the processing times keep unchanged. Then schedule the jobs by Johnson’s rule for \(F_2|C_{\text{max}}\), and denote the makespan as \(C^1_{\text{max}}\ (C^2_{\text{max}})\). We have \(C^J_{\text{max}} \leq \max\{C^1_{\text{max}}, C^2_{\text{max}}\}\).

The proof of lemma 1 is also given in Appendix B.

Theorem 11. Given \(\epsilon > 0\), the JJAR algorithm is a \((\frac{3}{2} + \epsilon)\)-approximation algorithm for \(J_2|op \leq 2, \text{shortest path}|C_{\text{max}}\).
Finally, we study the general case $J_m|\text{shortest path}|C_{\max}$, where $m$ is fixed. By theorem 5, we know that there exists $\alpha > 0$, such that the SSW-SSS algorithm returns a schedule satisfies

$$C'_\max \leq \alpha \frac{\log^2(m\mu)}{\log \log(m\mu)} \left( \max_{i \in \{1, \ldots, m\}} \sum_{j \in J'} \mu_{ij} p_{ij} + \max_{j \in J'} \sum_{i=1}^{m} \mu_{ij} p_{ij} \right).$$

The factor $\alpha$ is decided by choosing the probability of the randomized steps and the subsequent operations in the SSW-SSS algorithm [14, 16], and its value can be obtained by complicated calculation. Assume we determine such value of $\alpha$. We can design an approximation algorithm with worst-case ratio $O\left(\frac{\log^2(m\mu)}{\log \log(m\mu)}\right)$ for $J_m|\text{shortest path}|C_{\max}$. We refer to this algorithm as the SAR algorithm, and describe it in Algorithm 6.

**Algorithm 6 The SAR for $J_m|\text{shortest path}|C_{\max}$**

1: Set Alg be the SSW-SSS algorithm for $J_m|C_{\max}$ and $\rho = \frac{\log \log(m\mu)}{2\alpha \log^2(m\mu)}$.
2: Solve the problem by implementing UAR(Alg, $\rho$, $m$).

**Theorem 12.** The SAR algorithm is an $O\left(\frac{\log^2(m\mu)}{\log \log(m\mu)}\right)$-approximation algorithm for $J_m|\text{shortest path}|C_{\max}$.

However, we remind that the SAR algorithm relies on the assumption, that we can determine the constant $\alpha$ for the SSW-SSS algorithm. We can calculate it by following the details of the SSW-SSS algorithm, and in fact we can choose $\alpha$ large enough to guarantee the performance ratio of our algorithm.

### 4.3 A PTAS for $O_m|\text{shortest path}|C_{\max}$

In the previous subsection, we introduced a $(2 + \epsilon)$-approximation algorithm for $O_m|\text{shortest path}|C_{\max}$ based on the UAR(Alg, $\rho$, $m$) algorithm. By a different approach, we propose a $(1 + \epsilon)$-approximation algorithm for any $\epsilon > 0$, i.e. a PTAS. We also iteratively find feasible solutions, but guarantee that one of the returned solutions has the same first $N$-th largest jobs with an optimal solution where $N$ is a given constant. Precisely speaking, we say job $J_j$ is larger than job $J_k$ if $\max_{i \in \{1, \ldots, m\}} p_{ij} > \max_{i \in \{1, \ldots, m\}} p_{ik}$. To do this, we enumerate all size $N$ subsets $J^N$ of $J$, and then iteratively modify the weights of the graph such that the jobs larger than any job in $J^N$ will not be chosen. Then find a feasible solution which contains all the jobs in $J^N$ corresponding to the modified graph, i.e., the corresponding path is constrained to visit all the arcs corresponding to $J^N$ if such a path exists.

To find a feasible solution in each iteration, we adopt the modified ABV algorithm (see Appendix A) to obtain a near optimal min-max shortest path
among all the paths visiting the arcs corresponding to $J^N$ if such a path exists. Then we schedule the selected jobs by the PTAS for $Om||C_{\max}$ \cite{15} which is denoted as the SW algorithm. We refer to our algorithm as the SAE algorithm, and describe it in Algorithm 7.

Algorithm 7 The SAE algorithm for $Om|\text{shortest path}|C_{\max}$

1: Given $0 < \epsilon < 1$, set $N = m \left( \frac{m(3+\epsilon)}{1+\epsilon} \right)^{\frac{1}{2}}$.
2: Set $D := \emptyset$, $M := (1 + \frac{\epsilon}{3}) \sum_{j \in J} \sum_{i=1}^m p_{ij} + 1$.
3: Initially, $(w^1_j, w^2_j, \ldots, w^m_j) := (p_{1j}, p_{2j}, \ldots, p_{mj})$, for $a_j \in A$ corresponding to $J_j$.
4: for all $J^N \subseteq J$, with $|J^N| = N$ do
5: $(w^1_j, w^2_j, \ldots, w^m_j) := (p_{ij}, p_{2j}, \ldots, p_{mj})$, $D := \emptyset$.
6: For jobs $J_k \in J \setminus J^N$ with $\max_{i \in \{1, \ldots, m\}} p_{ik} > \min_{j \in J^N} \max_{i \in \{1, \ldots, m\}} p_{ij}$, set $(w^1_k, w^2_k, \ldots, w^m_k) := (M, M, \ldots, M)$, $D := D \cup \{J_k\}$.
7: Implement the modified ABV algorithm to obtain a feasible path $P$ of $SP$ such that the returned path visits all the arcs corresponding to $J^N$ if such a path exists. Construct the corresponding job set as $J_P$.
8: Schedule the jobs of $J_P$ by the SW algorithm.
9: if $C'_{\max} < C_{\max}$ then
10: $S := J_P$, $\sigma := \sigma'$, $C_{\max} := C'_{\max}$.
11: end if
12: end for
13: return $S$, $\sigma$, $C_{\max}$.

There are $\binom{n}{N}$ distinct subsets $J^N$, thus the iterations between line 4 - line 12 run at most $O(n^N)$ times, that is a polynomial of $n$ since $N$ is a constant when $m$ and $\epsilon$ are fixed. Since the modified ABV algorithm is a FPTAS and the SW algorithm is a PTAS, the running time of each iteration is also bounded by the polynomial of $n$ if $m$ and $\epsilon$ are fixed. It suffices to show that the SAE algorithm terminates in polynomial time. The following theorem indicates the SAE algorithm is a PTAS, and detailed proof can be found in Appendix C.

Theorem 13. The SAE algorithm is a PTAS for $Jm|\text{shortest path}|C_{\max}$.

5 Conclusions

This paper studies several problems combining two well-known combinatorial optimization problems. We show the hardness of the problems, and present some approximation algorithms. It is interesting to find approximation algorithms with better worst-case ratios for $J2|op \leq 2, \text{shortest path}|C_{\max}$ and $Jm|\text{shortest path}|C_{\max}$. Moreover, it needs further study to close the gap between the 2-inapproximability results and the $m$-approximation algorithms for $O|\text{shortest path}|C_{\max}$ and $J|\text{shortest path}|C_{\max}$. One can also consider other interesting combination of combinatorial optimization problems.
Acknowledgments

This work has been supported by the Bilateral Scientific Cooperation Project BIL10/10 between Tsinghua University and KU Leuven.

References

1. Aissi, H., Bazgan, C., Vanderpooten, D.: Approximating min-max (regret) versions of some polynomial problems. In: Chen, D., Pardalos, P.M. (eds.) COCOON 2006, LNCS, vol. 4112, pp. 428–438. Springer, Heidelberg (2006)
2. Bárány, I., Fiala, T.: Többgépes ütemezési problémák közel optimális megoldása (in Hungarian). Szigma - Matematikai - Közgazdasági Folyóirat 15, 177–191 (1982)
3. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271 (1959)
4. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, San Francisco (1979)
5. Gonzalez, T., Sahni, S.: Open shop scheduling to minimize finish time. Journal of the Association for Computing Machinery 23, 665–679 (1976)
6. Jackson, J.R.: An extension of Johnson’s results on job-lot scheduling. Naval Research Logistics Quarterly 3, 201–203 (1956)
7. Jansen, K., Solis-Oba, R., Srividenko, M.: Makespan minimization in job shops: A linear time approximation scheme. SIAM Journal on Discrete Mathematics 16, 288–300 (2003)
8. Johnson, S.M.: Optimal two- and three-stage production schedules with setup times included. Naval Research Logistics Quarterly 1, 61–68 (1954)
9. Lenstra, J.K., Kan, A.R., Brucker, P.: Complexity of machine scheduling problems. Annals of Operations Research 1, 343–362 (1977)
10. Lenstra, J., Rinnooy Kan, A.: Computational complexity of discrete optimization problems. Annals of Operations Research 4, 121–140 (1979)
11. Mastrolilli, M., Svensson, O.: Hardness of approximating flow and job shop scheduling problems. Journal of the Association for Computing Machinery 58, 20:1–20:32 (2011)
12. Nip, K., Wang, Z.: Combination of two-machine flow shop scheduling and shortest path problems. In: Du, D.Z., Zhang, G. (eds.) COCOON 2013, LNCS, vol. 7936, pp. 680–687. Springer, Heidelberg (2013)
13. Nip, K., Wang, Z.: Combination of flow shop scheduling and shortest path problem, working paper. Tsinghua University (2013)
14. Schmidt, J.P., Siegel, A., Srinivasan, A.: Chernoff-hoeffding bounds for applications with limited independence. SIAM Journal on Discrete Mathematics 8, 223–250 (1995)
15. Sevastianov, S.V., Woeginger, G.J.: Makespan minimization in open shops: A polynomial time approximation scheme. Mathematical Programming 82, 191–198 (1998)
16. Shmoys, D.B., Stein, C., Wein, J.: Improved approximation algorithms for shop scheduling problems. SIAM Journal on Computing 23, 617–632 (1994)
17. Wang, Z., Cui, Z.: Combination of parallel machine scheduling and vertex cover. Theoretical Computer Science 460, 10–15 (2012)
18. Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevast’janov, S.V., Shmoys, D.B.: Short shop schedules. Operations Research 45, 288–294 (1997)
Appendix A  The Modified ABV Algorithm

In this appendix, we give a modified version of the ABV algorithm [1]. The algorithm can determine a near optimal min-max shortest path among all the paths which are constrained to visit all arcs in a arc set $A' \subset A$ ($|A'| = N$ is a constant) if such a path exists. We propose a dynamic programming to solve this problem in pseudo-polynomial time.

We index the vertex set $V$ as $\{0, 1, \cdots, t\}$, where 0 is the starting point and $t$ is the destination. We denote $A' = \{a_1, a_2, \cdots, a_N\}$. For $u, v \in \{0, 1, \cdots, t\}$, let $S^u_{0}$ be a set of $(K + N)$-dimensional vectors, where $K$ is the number of weights of each arc. A vector $s(P) \in S^u_{0}$ with respect to a 0-v path $P$ with at most $u$ arcs, satisfies that its first $K$ components are the lengths of $P$ from 0 to $v$ with respect to different weights respectively, and each of the last $N$ components is associated with an arc in $A'$ such that the $(K + i)$-th component is 1 if $P$ visits $a_i$, and is 0 otherwise.

Notice that each of the first $K$ components of $s(P)$ can be bounded by $W = \max_{k \in \{1, \cdots, K\}} \sum_{j=1}^{n} w^k_j$ and the other $N$ components are binary, and thus the size (numbers of distinct vectors) of $S^u_{0}$ is no more than $W K 2^N = O(W K)$.

Algorithm 8 The Modified ABV Algorithm

```plaintext
1: $W = \max_{k \in \{1, \cdots, K\}} \sum_{j=1}^{n} w^k_j$.
2: $S^u_{0} := \{(0, 0, \cdots, 0)\}$, $S^u_{v} := \emptyset$ for $u, v \in \{0, 1, \cdots, t\}$.
3: for $u = 1, \cdots, t$ do
4:    for $v = 1, \cdots, t$ do
5:        for each $v'$ with $a_j = (v', v) \in A$ do
6:            for each vector $s(P) = (a^{1}, a^{2}, \cdots, a^{K+1}, a^{K+2}, \cdots, a^{K+N}) \in S^{v'-1}_{v'}$ do
7:                if $\forall i, a^{i} + w^{j}_i \leq W$ then
8:                    if $a_j \in A'$ then
9:                        $S^u_{v} := S^u_{v} \cup \{s(P) + s^0_j\}$.
10:                   else
11:                        $S^u_{v} := S^u_{v} \cup \{s(P) + s^1_j\}$.
12:                    end if
13:                end if
14:            end for
15:        end for
16:    end for
17: end for
18: return the path $P$ corresponding to a vector $(a^{1}, a^{2}, \cdots, a^{K}, a^{K+1}, \cdots, a^{N}) \in S^t_{t}$ such that $\max_{i=1, \cdots, K} a^{i}$ is the minimum among all such vectors if such a vector exists, and otherwise return an empty set.
```
It is not difficult to see that Algorithm 8 returns an optimal solution in time $O(|V|^2 |A| W^K)$, which is a pseudo-polynomial time algorithm. Based on the scaling technique (for example, see [1]), we can use Algorithm 8 to derive a FPTAS, such that given $\epsilon > 0$, it returns a path with the value at most $(1 + \epsilon) \min_{P} \max_{k=1, \ldots, K} \sum_{a_{j} \in P} w_{j}^k$ among all the paths $P$ visiting arc set $A'$ if such a path exists.

**Appendix B  The Performance Analysis of Algorithms in Section 4.2**

We point out that the proofs of the worst-case performance of algorithms based on UAR(Alg, $\rho$, $m$) are quite similar. We give the detailed proof for the GAR algorithm, and describe the key ideas and main steps for the other results since they can be obtained by analogous arguments.

**Proof (Theorem 9).** Let $C_{\text{max}}$ and $S$ be the makespan and the job set returned by the GAR algorithm respectively, and $C'_{\text{max}}$ and $J^*$ the value of an optimal solution. We consider two cases.

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

It implies that there is at least one job in the optimal solution, say $J_j$, such that $p_{1j} + p_{2j} \geq C'_{\text{max}}$ holds for a current schedule with makespan $C'_{\text{max}}$ during the execution. Notice that the schedule returned by the GAR algorithm is the best one among all current schedules, i.e. $C_{\text{max}} \leq C'_{\text{max}}$. It follows from (3) that

$$C_{\text{max}} \leq C'_{\text{max}} \leq p_{1j} + p_{2j} \leq C^*_{\text{max}}.$$  

That is to say the GAR algorithm will return an optimal solution for this case.

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

Consider the last current schedule during the execution of the GAR algorithm. We denote the corresponding job set and the makespan as $J'$ and $C'_{\text{max}}$ respectively.

In this case, we first argue that $J' \cap D = \emptyset$. Suppose that this is not the case, since $J^* \cap D = \emptyset$, the weights of arcs corresponding to the jobs in $J^*$ have not been revised. Hence we have $(1 + \epsilon) \max \left\{ \sum_{J_j \in J} w_j^1, \sum_{J_j \in J^*} w_j^2 \right\} < M$. Moreover, by the assumption $J' \cap D \neq \emptyset$, we have $\max \left\{ \sum_{J_j \in J'} w_j^1, \sum_{J_j \in J^*} w_j^2 \right\} \geq M$. By Theorem 6, the solution returned by the ABV algorithm satisfies

$$M \leq \max \left\{ \sum_{J_j \in J'} w_j^1, \sum_{J_j \in J^*} w_j^2 \right\} \leq (1 + \epsilon) \max \left\{ \sum_{J_j \in J'} w_j^1, \sum_{J_j \in J^*} w_j^2 \right\} < M,$$

which leads to a contradiction.
Notice that each job in the last current schedule satisfies $p_{1j} + p_{2j} < C'_\text{max}$, since otherwise the algorithm will continue. Therefore, by Theorem 1 the schedule returned by the GS algorithm satisfies

$$C'_\text{max} = \max \left\{ \sum_{j \in J'} p_{1j}, \sum_{j \in J'} p_{2j} \right\},$$

(6)

Since $J' \cap D = \emptyset$, we know that all jobs $J_j \in J'$ have not been revised. Thus, it follows from (1), (3), (6), Theorem 6 and the fact that the schedule returned by the GAR algorithm is the best one among all current schedules, we have

$$C_{\text{max}} \leq C'_\text{max} = \max \left\{ \sum_{j \in J'} p_{1j}, \sum_{j \in J'} p_{2j} \right\}$$

$$\leq (1 + \epsilon) \max \left\{ \sum_{j \in J^*} w_{1j}, \sum_{j \in J^*} w_{2j} \right\}$$

$$\leq (1 + \epsilon) C^*_{\text{max}}.$$

We have claimed that the algorithms based on UAR($\text{Alg}, \rho, m$) are polynomial-time algorithms. Moreover, notice that the ABV algorithm is a FPTAS, the GS algorithm runs in linear time, and the GAR algorithm implements the ABV algorithms and the GS algorithm at most $n$ times, and we can claim that the GAR algorithm is a FPTAS for $O_2|\text{shortest path}|C_{\text{max}}$.

In the following proofs of Theorems 10, 11 and 12, we adopt the same notations as in the proof of Theorem 9, and also analyze the same two cases. Then we give the proof of the RAR algorithm for $O_m|\text{shortest path}|C_{\text{max}}$.

**Proof (Theorem 10).** The argument for the first case is similar to that of Theorem 9 by noticing that there is at least one job with $\sum_{i=1}^{m} p_{ij} > \frac{1}{2} C'_\text{max}$ in both the optimal schedule and one current schedule, and it follows that $C_{\text{max}} \leq C'_\text{max} \leq 2 C^*_{\text{max}}$.

For the second case, notice that by Theorem 2 the makespan of the last current schedule returned by Ráczmány algorithm satisfies $C'_{\text{max}} \leq \sum_{J_j \in J'} p_{ij} + \sum_{i=1}^{m} p_{ik}$, where $J_k$ is the last completed job and processed on $M_k$. Moreover, each jobs in $J'$ satisfies $\sum_{i=1}^{m} p_{ij} \leq \frac{1}{2} C'_\text{max}$, as otherwise the algorithm will continue. By Theorem 6 and a similar argument as in Theorem 9 it is not difficult to show that $C_{\text{max}} \leq C'_\text{max} \leq (2 + \epsilon) C^*_{\text{max}}$.

Before analyzing the performance of $J2|o \text{p} \leq 2, \text{shortest path}|C_{\text{max}}$, we first prove Lemma 1.
Denote $J_{12}$ ($J_{21}$) as the set of jobs with processing order $(1 \to 2)$ ($(2 \to 1)$) in the original job set. In the schedule returned by Jackson’s rule for $J2|op \leq 2|C_{\text{max}}$, let $C^J_{\text{max}}$ be the makespan, and suppose that the total processing time of $J_{12}$ on $M_1$ is not less than that of $J_{21}$ on $M_2$. By Jackson’s rule, jobs in $J_{21}$ are scheduled after jobs in $J_{12}$ consecutively on $M_1$, therefore no idle occurs on $M_1$. We consider the following cases.

**Case 1.** $C^J_{\text{max}} = \sum_{j \in J} p_{ij}$

It follows from [1] that $C^J_{\text{max}} \leq \max\{C^1_{\text{max}}, C^2_{\text{max}}\}$.

**Case 2.** $C^J_{\text{max}} > \sum_{j \in J} p_{ij}$

**Subcase 2.1.** no idle occurs on $M_2$

In this case, the processes on both machines are consecutive, it is straightforward to show that $C^J_{\text{max}} = \sum_{j \in J} p_{ij} \leq \max\{C^1_{\text{max}}, C^2_{\text{max}}\}$.

**Subcase 2.2.** idle occurs on $M_2$

Remember that Jackson’s rule first schedules jobs in $J_{12}$ and $J_{21}$ by Johnson’s rule respectively, denoted the two schedules as $\sigma_1$ and $\sigma_2$, and then combines the two schedules. Since Johnson’s rule returns a permutation schedule, we can denote $\sigma_1 = \{1, \cdots, l\}$ and $J_{12} = \{J_1, \cdots, J_l\}$. Consider the job in $J_{12}$, say $J_k$, which starts the processing on $M_2$ after the last idle on that machine. It is easy to see that $J_k$ starts processing on $M_2$ immediately after its completion on $M_1$. Recall that no idle occurs on $M_1$ by assumption, thus we have $C^J_{\text{max}} = \sum_{j=1}^k p_{1j} + \sum_{j=k}^l p_{2j}$. Notice that $\sigma_1$ is obtained by Johnson’s rule. We change all jobs’ processing order as $(1 \to 2)$ and obtain a schedule by applying Johnson’s rule to the revised jobs. Let the makespan of this schedule be $C^1_{\text{max}}$. Since this schedule is also obtained by Johnson’s rule, we know $J_1, \cdots, J_{k-1}$ are also scheduled before $J_k$, whereas $J_{k+1}, \cdots, J_l$ are scheduled after $J_k$ in this schedule. Therefore, we have $C^1_{\text{max}} \geq \sum_{j=1}^k p_{1j} + \sum_{j=k}^l p_{2j}$, and it suffices to show that $C^J_{\text{max}} \leq C^1_{\text{max}}$.

For the case where total processing time of $J_{12}$ on $M_1$ is less than that of $J_{21}$ on $M_2$, an analogous argument also yields $C^J_{\text{max}} \leq \max\{C^1_{\text{max}}, C^2_{\text{max}}\}$. □

Now we can study the performance of the JJAR algorithm for $J2|op \leq 2, \text{shortest path}|C_{\text{max}}$.

**Proof (Theorem 11).** The first case is similar to that of Theorem 3 by noticing that there is at least one job with $\sum_{i=1}^m p_{ij} > \frac{4}{3}C^J_{\text{max}}$ in both the optimal schedule and one current schedule, it follows that $C_{\text{max}} \leq C^J_{\text{max}} \leq \frac{3}{2}C^J_{\text{max}}$.

For the second case, first by lemma 1 we have $C^J_{\text{max}} \leq \max\{C^1_{\text{max}}, C^2_{\text{max}}\}$, where $C^J_{\text{max}}$ is the makespan of the last current schedule, and $C^J_{\text{max}}$ ($C^2_{\text{max}}$) is the makespan of the schedule obtained by changing the processing order of all jobs of $J'$ to be $(1 \to 2)$ ($(2 \to 1)$) and applying Johnson’s rule. Assume that $C^J_{\text{max}} \leq C^J_{\text{max}}$, and denote $J_v$ as the critical job of the schedule with respect to $C^J_{\text{max}}$. If $p_{1v} \geq p_{2v}$, we have $C^J_{\text{max}} \leq \sum_{j \in J'} p_{1j} + p_{2v}$. Notice that all jobs in the last current schedule satisfy $p_{1j} + p_{2j} \leq \frac{4}{3}C^J_{\text{max}}$ in the JJAR algorithm,
and we have $p_{2v} \leq \frac{1}{3} C_{\text{max}}'$. A similar argument as in Theorem 9 shows that $C_{\text{max}} \leq C_{\text{max}}' \leq C_{\text{max}}^* \leq (\frac{3}{2} + \epsilon) C_{\text{max}}^*$. The other situations can be verified by analogous arguments. Thus, the JJAR algorithm is $(\frac{3}{2} + \epsilon)$-approximate.

Finally, we give the proof of the SAR algorithm for $Jm|\text{shortest path}|C_{\text{max}}$.

Proof (Theorem 12). The first case is analogous, and we can show that $C_{\text{max}} \leq \frac{1}{\log \log(m \mu)} C_{\text{max}}^* \leq \frac{1}{\log \log(m \mu)} C_{\text{max}}^\frac{1}{\log \log(m \mu)} C_{\text{max}}^*$.

For the second case, all the jobs in $J'$ satisfy

$$\sum_{i=1}^{m} \mu_{ij} p_{ij} \leq \frac{\log \log(m \mu)}{2x \log \log(m \mu)} C_{\text{max}}^*,$$

as otherwise the algorithm will continue. Combining (4) and Theorem 6, by a similar argument as in Theorem 9, it is not difficult to show that $C_{\text{max}} \leq \frac{1}{\log \log(m \mu)} C_{\text{max}}^*$. Thus, there exists an $O((\log \log(m \mu))/\log \log(m \mu))$-approximation algorithm for this problem.

Appendix C The Proof of Theorem 13 in Section 4.3

This appendix analyzes the performance of the SAE algorithm.

Proof (Theorem 13). Remember that we have assumed $N$ is a constant.

Consider the iteration that the subset $J^N$ is exactly the first $N$-th largest jobs of $J^*$, and denote the makespan and the job set returned in this iteration as $C_{\text{max}}'$ and $J'$ respectively.

We now argue that the jobs in $J^N$ are also the first $N$-th largest jobs of $J'$. First, the modified ABV algorithm returns a path visiting the arcs corresponding to $J^N$ if such a path exists. Since $J^N$ is exactly the first $N$-th largest jobs of $J^*$, we have $J^* \cap D = \emptyset$ and $J' \cap D = \emptyset$ following the analogous arguments in the proof of the algorithms based on UAR($\text{Alg}, \rho, m$). Therefore $J^N$ is exactly the set of first $N$-th largest jobs of $J'$. Notice that $C_{\text{max}}$ is the best one among all current schedules, and we have $C_{\text{max}} \leq C_{\text{max}}'$, so we only concern about $C_{\text{max}}'$ and the schedule returned in that iteration in the subsequent analysis.

Denote $P_{\text{max}}' = \max_{i \in \{1, \ldots, m\}} \max_{i \in J^*} \sum_{a_j \in J^*} p_{ij}$. In the SAE algorithm, given $\epsilon > 0$, we can use the modified ABV algorithm to return a path satisfying

$$P_{\text{max}}' \leq \left(1 + \frac{\epsilon}{3}\right) \max_{i \in \{1, \ldots, m\}} \sum_{a_j \in J^*} p_{ij},$$

thus from (1) we have,

$$P_{\text{max}}' \leq \left(1 + \frac{\epsilon}{3}\right) C_{\text{max}}^*.$$  

Now we study the schedule returned by the SW algorithm. Recall that jobs are divided into large jobs and small jobs [13]:

$$J_L' = \{J_j \in J'| p_{ij} \geq \alpha P_{\text{max}}', \text{ for some } i, 1 \leq i \leq m\},$$

$$J_S' = \{J_j \in J' \setminus J_L'\}.$$
Furthermore, the operations of jobs in $J'_S$ are divided into two sets:

$$O_1^S = \{ O_{ij} | \alpha^2 P'_{\max} < p_{ij} \leq \alpha P'_{\max}, \; J_j \in J'_S \},$$

$$O_2^S = \{ O_{ij} | p_{ij} \leq \alpha^2 P'_{\max}, \; J_j \in J'_S \}.$$

The value of $\alpha$ is determined by the inequalities

$$\left( \frac{2\alpha}{m(3+\epsilon)} \right)^{m(3+\epsilon)} < \alpha \leq \frac{\epsilon}{m(3+\epsilon)}.$$  \hspace{1cm} (9)$$

and

$$\sum_{O_{ij} \in O_1^S} p_{ij} \leq \frac{\epsilon}{3+\epsilon} P'_{\max}.$$  \hspace{1cm} (10)

We show that such $\alpha$ exists and can be found in polynomial time. Denote $\alpha_k = \left( \frac{\epsilon}{m(3+\epsilon)} \right)^{2k}$, $k = 0, 1, \ldots, \frac{m(3+\epsilon)}{\epsilon} - 1$, and $O_1^S(\alpha_k)$ as the operations of $J'_S$ by setting $\alpha = \alpha_k$. Thus we have $\frac{m(3+\epsilon)}{\epsilon}$ disjoint operation sets $\{ O_1^S(\alpha_k) \}$. Notice that the total processing time of all the operations is at most $mP'_{\max}$, thus there must be at least one $O_1^S(\alpha_k)$ satisfying (10), then we set $\alpha$ be such $\alpha_k$. Such $\alpha$ can be found in constant time for fixed $m$ and $\epsilon$.

Notice that the number of large jobs satisfy $|J'_L| \leq \frac{m}{\alpha_k} \leq m \left( \frac{\epsilon}{m(3+\epsilon)} \right)^{m(3+\epsilon)} = N$. Consequently, all the large jobs of $J'_L$ belongs to the job set $J^N$, and thus belongs to $J^*$. Moreover, notice that the SW algorithm first find an optimal schedule of $J'_L$ by trying any possible order, denote the makespan of this schedule as $C'_{\max}$. Recall that all jobs in $J^N$ also belong to $J^*$, therefore we have

$$C'^L_{\max} \leq C^*_{\max}. \hspace{1cm} (11)$$

The remaining analysis is based on the SW algorithm \[15\].

Let $M_l$ be the last completed machine, and $t$ be the completion time of the last operation of jobs in $J'_L$ on $M_l$. $T_0$ and $T_t$ are referred to the idle time on $M_l$ during time intervals $[0,t]$ and $[t, C'_{\max}]$ respectively. The total processing time of all operations on $M_l$ after time $t$ is denoted by $P_t$. The worst worst-case ratio is shown by considering several cases as follows.

Case 1. $P_t = 0$

It implies that $C'_{\max} = t \leq C'^L_{\max}$. By (11), the algorithm returns an optimal schedule.

Case 2. $P_t \neq 0$

Consider the last operation $O_{ij}$ on $M_l$. By assumption, job $J_j$ belongs to $J'_S$. Since $J_j$ cannot be processed on any other idles on $M_l$ after time $t$, at these idles there must be some machine processing $J_j$. By combining (8) and (9), we have

$$T_t \leq \sum_{i \neq l} p_{ij} \leq (m-1) \alpha P'_{\max} \leq \frac{\epsilon}{3+\epsilon} P'_{\max} \leq \frac{\epsilon}{3} C^*_{\max}. \hspace{1cm} (12)$$
Let $O'$ be the set of $O_S^2$ processed on $M_l$ after $t$. We consider the following subcases.

**Subcase 2.1.** $|O'| \leq m - 1$

By (9), (10) and (8), the total processing time on $M_l$ after time $t$ (the jobs are all in $J_S^l$) satisfies

$$P_t \leq \sum_{O_{ij} \in O_y} p_{ij} + \sum_{O_{ij} \in O'} p_{ij} \leq \frac{\epsilon}{3 + \epsilon} P'_{\max} + (m - 1)\alpha^2 P'_{\max} \leq \frac{2}{3} \epsilon C_{\max}^*.$$  \hfill (13)

Thus, it follows from (11), (12) and (13) that

$$C_{\max} \leq C_{\max}' = t + P_t + T_t \leq C_{\max}^L + P_t + T_t \leq C_{\max}^* + \frac{2}{3} \epsilon C_{\max}^* + \frac{\epsilon}{3} C_{\max} = (1 + \epsilon)C_{\max}^*.$$  \hfill (14)

**Subcase 2.2.** $|O'| > m - 1$

Notice that there are at most $m/\alpha$ idles (large jobs) before $t$ on machine $M_l$ in the schedule with respect to $C_{\max}'$. On each idle, since there are at most $m - 1$ jobs being processed on the other machines and $|O'| > m - 1$, the idle must smaller than $\alpha^2 P'_{\max}$, since otherwise some job will be scheduled on that idle. Therefore we have

$$T_0 \leq \frac{m}{\alpha} \alpha^2 P^*_{\max} \leq \frac{\epsilon}{3 + \epsilon} P'_{\max} \leq \frac{\epsilon}{3} C_{\max}^*.$$  \hfill (15)

Since $C_{\max}'$ can be written as $C_{\max}' = \sum_{j \in J'} p_{tj} + T_0 + T_t$, it follows from (8), (12) and (15) that

$$C_{\max} \leq C_{\max}' = \sum_{j \in J'} p_{tj} + T_0 + T_t \leq P'_{\max} + T_0 + T_t \leq (1 + \frac{\epsilon}{3})C_{\max}^* + \frac{2}{3} \epsilon C_{\max} = (1 + \epsilon)C_{\max}^*.$$  \hfill (16)

In conclusion, the SAE algorithm produces a schedule with makespan at most $(1 + \epsilon)C_{\max}^*$.  \qed