Journal of Combinatorial Optimization (2022) 43:571–588
https://doi.org/10.1007/s10878-021-00793-3

Path cover with minimum nontrivial paths and its application in two-machine flow-shop scheduling with a conflict graph

Yong Chen[1] · Yinhui Cai[1] · Longcheng Liu[2] · Guangting Chen[3] · Randy Goebel[4] · Guohui Lin[4] · Bing Su[5] · An Zhang[1]

Accepted: 2 August 2021 / Published online: 16 August 2021

Abstract
Path cover is a well-known intractable problem that finds a minimum number of vertex disjoint paths in a given graph to cover all the vertices. We show that a variant, in which the objective is to minimize the number of length-0 paths, is polynomial-time solvable. We further show that another variant, to minimize the total number of length-0 and length-1 paths, is also polynomial-time solvable. Both variants find applications in approximating the two-machine flow-shop scheduling problem in which job processing has constraints that are formulated as a conflict graph. For the unit jobs, we present a $4/3$-approximation for the scheduling problem with an arbitrary conflict graph, based on the exact algorithm for the above second variant of the path cover problem. For arbitrary jobs where the conflict graph is the union of two disjoint cliques, we present a simple $3/2$-approximation algorithm.

Keywords Path cover · Flow-shop scheduling · Conflict graph, *b*-matching · Approximation algorithm

1 Introduction

Scheduling is a well established research area that finds numerous applications in modern manufacturing industry and in operations research at large. All scheduling problems modeling real-life applications have at least two components, the machines and the jobs. In one big category of problems that have received intensive studies, scheduling constraints are imposed between a machine and a job, such as a time interval during which the job is allowed to be processed *nonpreemptively* on the machine,

Yong Chen, Yinhui Cai, Longcheng Liu are co-first authors.

An extended abstract appears in *Proceedings of COCOON 2018* Cai et al. (2018).

Extended author information available on the last page of the article while the machines are considered as independent from each other, so are the jobs. For example, the parallel machine scheduling (the *multiprocessor scheduling* in Garey and Johnson (1979)) is one of the first studied problems, denoted as $Pm \mid\mid C_{\max}$ in the three-field notation (Graham et al. 1979), in which each job needs to be processed by one of the m given identical machines, with the goal to minimize the maximum job completion time (called the *makespan*); the m-machine flow-shop scheduling (the *flow-shop scheduling* in Garey and Johnson (1979)) is another first-studied problem, denoted as $Fm \mid\mid C_{\max}$, in which each job needs to be processed by all the m given machines in the same sequential order, with the goal to minimize the makespan.

In another category of scheduling problems, additional but limited resources are required for the machines to process the jobs (Garey and Graham 1975). The resources are renewable but normally non-sharable in practice; the jobs competing for the same resource have to be processed at different time if their total demand for a certain resource exceeds the supply. Scheduling with resource constraints (Garey and Graham 1975; Garey and Johnson 1975) or scheduling with conflicts (SwC) (Even et al. 2009) also finds numerous applications (Bodlaender and Jansen 1995; Baker and Coffman 1996; Halldórsson et al. 2003) and has attracted as much attention as the non-constrained counterpart. In this paper, we use SwC to refer to the nonpreemptive scheduling problems with additional constraints or conflicting relationships among the jobs to disallow them to be processed concurrently on different machines. We remark that in the literature, SwC is also presented as the scheduling with agreements (SwA), in which a subset of jobs can be processed concurrently on different machines if and only if they are agreeing with each other (Bendraouche and Boudhar 2012, 2016). While in the most general scenario a conflict could involve multiple jobs, in this paper we consider only those conflicts each involves two jobs and consequently all the conflicts under consideration can be presented as a *conflict graph* $G = (V, E)$, where V is the set of jobs and an edge $e = (J_{j_1}, J_{j_2}) \in E$ represents a conflicting pair such that the two jobs J_{j_1} and J_{j_2} cannot be processed concurrently on different machines in any feasible schedule.

Extending the three-field notation (Graham et al. 1979), the parallel machine SwC with a conflict graph $G = (V, E)$ (also abbreviated as SCI in the literature) (Even et al. 2009) is denoted as $Pm \mid G = (V, E), p_j \mid C_{\max}$, where the first field Pm tells that there are m parallel identical machines, the second field describes the conflict graph $G = (V, E)$ over the set V of all the jobs, where the job J_j requires a non-preemptive processing time of p_j on any machine, and the last field specifies the objective function to minimize the makespan $C_{\max}$. One clearly sees when $E = \emptyset$, $Pm \mid G = (V, E), p_j \mid C_{\max}$ reduces to the classical multiprocessor scheduling $Pm \mid\mid C_{\max}$, which is already NP-hard for $m \geq 2$ (Garey and Johnson 1979). Indeed, with m either a given constant or part of input, $Pm \mid G = (V, E), p_j \mid C_{\max}$ is more difficult to approximate, and there is a line of rich research to consider the unit jobs (that is, $p_j = 1$) and/or to consider certain special classes of conflict graphs. The interested reader might refer to Even et al. (2009) and the references therein.

In this paper, we are interested in the two-machine flow-shop SwC. Note that in the general m-machine (also called m-stage) flow-shop (Garey and Johnson 1979) denoted as $Fm \mid\mid C_{\max}$, there are $m \geq 2$ machines $M_1, M_2, \ldots, M_m$, a set V of jobs, each job J_j needs to be processed through $M_1, M_2, \ldots, M_m$ sequentially with processing times

$p_{1j}, p_{2j}, \ldots, p_{mj}$ respectively. When $m = 2$, the two-machine flow-shop problem is polynomial time solvable, by Johnson's algorithm (Johnson 1954); the m-machine flow-shop problem when $m \geq 3$ is *strongly* NP-hard (Garey et al. 1976). After several efforts (Johnson 1954; Garey et al. 1976; Gonzalez and Sahni 1978; Chen et al. 1996), Hall presented a polynomial-time approximation scheme (PTAS) for the m-machine flow-shop problem, for any fixed integer $m \geq 3$ (Hall 1998). When m is part of input (*i.e.* an arbitrary integer), there is no known constant ratio approximation algorithm, and the problem cannot be approximated within 1.25 unless P = NP (Williamson et al. 1997).

The m-machine flow-shop SwC was first studied in 1980's. Blazewicz et al. (1983) considered multiple resource characteristics including the number of resource types, resource availabilities and resource requirements; they expanded the middle field of the three-field notation to express these resource characteristics, for which the conflict relationships are modeled by complex structures such as hypergraphs. At the end, they proved complexity results for several variants in which either the conflict relationships are simple enough or only the unit jobs are considered. Further studies on more variants can be found in Röck (1983, 1984); Błażewicz et al. (1988); Süral et al. (1992). In this paper, we consider those conflicts each involves only two jobs such that all the conflicts under consideration can be presented as a conflict graph $G = (V, E)$. The m-machine flow-shop scheduling with a conflict graph $G = (V, E)$ is denoted as $Fm \mid G = (V, E), p_{ij} \mid C_{\max}$. We remark that our notation is slightly different from the one introduced by Blazewicz et al. (1983), which uses a prefix "res" in the middle field for describing the resource characteristics.

Several applications of the m-machine flow-shop scheduling with a conflict graph were mentioned in the literature. In a typical example of scheduling medical tests in an outpatient health care facility where each patient (regarded as a job) needs to do a sequence of m tests (regarded as the machines), a patient must be accompanied by their doctor during a test and thus two patients under the care of the same doctor cannot go for tests simultaneously. That is, two patients of the same doctor are conflicting to each other, and all the conflicts can be effectively described as a graph $G = (V, E)$, where V is the set of all the patients and an edge represents a conflicting pair of patients.

In two recent papers, Tellache and Boudhar (2017; 2018) studied the problem $F2 \mid G = (V, E), p_{ij} \mid C_{\max}$, which they denoted as FSC. In Tellache and Boudhar (2018), the authors summarized and/or proved several complexity results; to name a few, $F2 \mid G = (V, E), p_{ij} \mid C_{\max}$ is strongly NP-hard when $G = (V, E)$ is the complement of a complete split graph (Tellache and Boudhar 2018; Blazewicz et al. 1983) (that is, G is the union of a clique and an independent set), $F2 \mid G = (V, E), p_{ij} \mid C_{\max}$ is weakly NP-hard when $G = (V, E)$ is the complement of a complete bipartite graph (Tellache and Boudhar 2018) (that is, G is the union of two disjoint cliques), and for an arbitrary conflict graph $G = (V, E)$, $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ is strongly NP-hard (Tellache and Boudhar 2018). In Tellache and Boudhar (2017), the authors proposed three mixed-integer linear programming models and a branch and bound algorithm to solve the last variant $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ exactly; their empirical study showed that the branch and bound algorithm outperforms and can solve instances of up to 20000 jobs.

In this paper, we pursue approximation algorithms with provable performance for the NP-hard variants of the two-machine flow-shop scheduling with a conflict graph. In Sect. 2, we present a $4/3$-approximation for the strongly NP-hard scheduling problem $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ for the unit jobs with an arbitrary conflict graph. In Sect. 3, we present a simple $3/2$-approximation for the weakly NP-hard scheduling problem $F2 \mid G = K_\ell \cup K_{n-\ell}, p_{ij} \mid C_{\max}$ for arbitrary jobs with a conflict graph that is the union of two disjoint cliques (that is, the complement of a complete bipartite graph). Some concluding remarks are provided in Sect. 4.

2 Approximating $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$

Tellache and Boudhar proved that $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ is strongly NP-hard by a reduction from the well known Hamiltonian path problem, which is strongly NP-complete (Garey and Johnson 1979). Furthermore, they remarked that $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ has a feasible schedule of makespan $C_{\max} = n + k$ if and only if the complement $\overline{G}$ of the conflict graph G, called the agreement graph, has a path cover of size k (that is, a collection of k vertex-disjoint paths that covers all the vertices of the graph $\overline{G}$), where n is the number of jobs (or vertices) in the instance. This way, $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ is polynomially equivalent to the Path Cover problem, which is NP-hard even on some special classes of graphs including planar graphs (Garey et al. 1976), bipartite graphs (Golumbic 2004), chordal graphs (Golumbic 2004), chordal bipartite graphs (Müller 1996) and strongly chordal graphs (Müller 1996).

In terms of approximability, to the best of our knowledge there is no $o(n)$-approximation for the Path Cover problem. Nevertheless, since a minimum path cover has size in between 1 and n, $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ admits a trivial 2-approximation algorithm. Furthermore, if there is an α-approximation algorithm for the Path Cover problem, where $\alpha \geq 1$, then it is also an $(\alpha + 1)/2$-approximation algorithm for $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$. However, one sees that our to-be presented $4/3$-approximation algorithm for $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ is only an $O(n)$-approximation algorithm for the Path Cover problem.

We give some terminologies first. The conflict graphs considered in this paper are all simple graphs. All paths and cycles in a graph are also simple. The number of edges on a path/cycle defines the length of the path/cycle. A length-k path/cycle is also called a k-path/cycle for short. Note that a single vertex is regarded as a 0-path, while a cycle has length at least 3. For an integer $b \geq 1$, a b-matching of a graph is a spanning subgraph in which every vertex has degree no greater than b; a maximum b-matching is a b-matching that contains the maximum number of edges. A maximum b-matching of a graph can be computed in $O(m^2 \log n \log b)$-time (specifically, a maximum 2-matching of a graph can be computed in $O(mn \log n)$-time), where n and m are the number of vertices and the number of edges in the graph, respectively (Gabow 1983). Clearly, a graph could have multiple distinct maximum b-matchings.

Given a graph, a path cover is a collection of vertex-disjoint paths in the graph that covers all the vertices, and the size of the path cover is the number of paths therein. The Path Cover problem is to find a path cover of a given graph of the minimum size, and the well known Hamiltonian path problem is to decide whether a given graph has a path cover of size 1. Besides the PATH COVER problem, many its variants have also been studied in the literature (Asdre and Nikolopoulos 2007; Pao and Hong 2008; Asdre and Nikolopoulos 2010; Rizzi et al. 2014). We mentioned earlier that Tellache and Boudhar proved that $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ is polynomially equivalent to the PATH COVER problem, but to the best of our knowledge there is no approximation algorithm designed for $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$. Nevertheless, one easily sees that, since $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ has a feasible schedule of makespan $C_{\max} = n+k$ if and only if the complement $\overline{G}$ of the conflict graph G has a path cover of size k, a trivial algorithm simply processing the jobs one by one (each on the first machine M_1 and then on the second machine M_2) produces a schedule of makespan $C_{\max} = 2n$, and thus is a 2-approximation algorithm.

In this section, we will design two approximation algorithms with improved performance ratios for $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$. These two approximation algorithms are based on our polynomial time exact algorithms for two variants of the PATH COVER problem, respectively. We start with the first variant called the *Path Cover with the minimum number of* 0*-paths*, in which we are given a graph and we aim to find a path cover that contains the minimum number of 0-paths. In the second variant called the *Path Cover with the minimum number of* $\{0, 1\}$*-paths*, we aim to find a path cover that contains the minimum total number of 0-paths and 1-paths. We remark that in both variants, we do not care about the size of the path cover.

2.1 Path Cover with the minimum number of 0-paths

Recall that in this variant of the PATH COVER problem, given a graph, we aim to find a path cover that contains the minimum number of 0-paths. The given graph is the complement $\overline{G} = (V, \overline{E})$ of the conflict graph $G = (V, E)$ in $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$. We next present a polynomial time algorithm that finds for $\overline{G}$ a path cover that contains the minimum number of 0-paths.

In the first step, we apply the $O(mn \log n)$-time algorithm (Gabow 1983) to find a maximum 2-matching in $\overline{G}$, denoted as M, where $n = |V|$ and $m = |\overline{E}|$. The 2-matching M is a collection of vertex-disjoint paths and cycles; let $\mathcal{P}_0$ ($\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_{\geq 3}$, $\mathcal{C}$, respectively) denote the sub-collection of 0-paths (1-paths, 2-paths, paths of length at least 3, cycles, respectively) in M. That is, $M = \mathcal{P}_0 \cup \mathcal{P}_1 \cup \mathcal{P}_2 \cup \mathcal{P}_{\geq 3} \cup \mathcal{C}$.

Clearly, if $\mathcal{P}_0 = \emptyset$, then we have a path cover containing no 0-paths after removing one edge per cycle in $\mathcal{C}$. In the following discussion we assume the existence of a 0-path, which is often called a *singleton*. We also call an ending vertex of a k-path with $k \geq 1$ as an endpoint for simplicity. The following lemma is trivial due to the edge maximality of M.

Lemma 2.1 *All the singletons and endpoints in the maximum* 2*-matching* M *are pairwise non-adjacent to each other in the underlying graph* $\overline{G}$.

Let v_0 be a singleton. If v_0 is adjacent to a vertex v_1 on a cycle of $\mathcal{C}$ in the underlying graph $\overline{G}$, then we may delete a cycle-edge incident at v_1 from M while adding the edge (v_0, v_1) to M to achieve another maximum 2-matching with one less singleton.

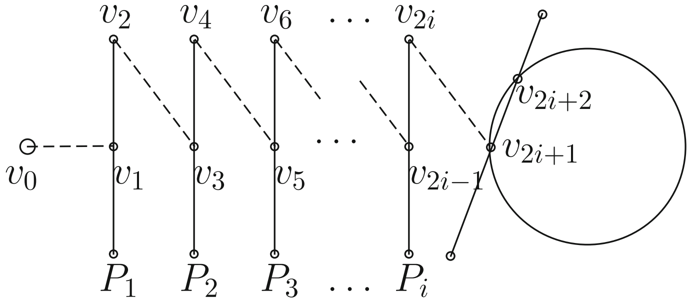

Fig. 1 An alternating path v_0-v_1-v_2-…-v_{2i}-v_{2i+1}-v_{2i+2} that saves the singleton v_0, where the last two vertices are on a cycle of $\mathcal{C}$ or a path of $\mathcal{P}_{\geq 3}$. In the figure, solid edges are in the maximum 2-matching M and dashed edges are outside of M

Similarly, if v_0 is adjacent to a vertex v_1 on a path of $\mathcal{P}_{\geq 3}$ (note that v_1 has to be an internal vertex on the path by Lemma 2.1) in the underlying graph $\overline{G}$, then we may delete a certain path-edge incident at v_1 from M while adding the edge (v_0, v_1) to M to achieve another maximum 2-matching with one less singleton. In either of the above two cases, assume the edge deleted from M is (v_1, v_2); then we say the *alternating* path v_0-v_1-v_2 *saves* the singleton v_0.

In the general setting, in the underlying graph $\overline{G}$, v_0 is adjacent to the middle vertex v_1 of a 2-path P_1, one endpoint v_2 of P_1 is adjacent to the middle vertex v_3 of another 2-path P_2, one endpoint v_4 of P_2 is adjacent to the middle vertex v_5 of another 2-path P_3, and so on, one endpoint v_{2i-2} of P_{i-1} is adjacent to the middle vertex v_{2i-1} of another 2-path P_i, one endpoint v_{2i} of P_i is adjacent to a vertex v_{2i+1} of a cycle of $\mathcal{C}$ or a path of $\mathcal{P}_{\geq 3}$ (see an illustration in Fig. 1), on which the edge (v_{2i+1}, v_{2i+2}) is to be deleted. Then we may delete the edges $\{(v_{2j+1}, v_{2j+2}) \mid j = 0, 1, \dots, i\}$ from M while add the edges $\{(v_{2j}, v_{2j+1}) \mid j = 0, 1, \dots, i\}$ to M to achieve another maximum 2-matching with one less singleton; and we say the *alternating* path v_0-v_1-v_2-…-v_{2i}-v_{2i+1}-v_{2i+2} *saves* the singleton v_0.

Lemma 2.2 *Given a maximum 2-matching M and a singleton v_0 therein, finding a simple alternating path to save v_0, if exists, can be done in $O(m)$ time, where $m = |\overline{E}|$.*

Proof Firstly, if an alternating path is not simple, then a cycle that forms a subpath is also alternating and has an even length, and thus the cycle can be removed resulting in a shorter alternating path. Repeating this process if necessary, at the end we achieve a simple alternating path. Therefore, we can limit the search for a simple alternating path.

Note that the edges on all possible alternating paths that save v_0 can be of the following four kinds: 1) all those edges incident at v_0, each oriented out of v_0; 2) all those edges of the 2-paths, each oriented from the middle vertex to the endpoint; 3) all those edges each connecting an endpoint of a 2-path to the middle vertex of another 2-path, oriented from the endpoint to the middle vertex; 4) all those edges each connecting an endpoint of a 2-path to a vertex on some path of $\mathcal{P}_{\geq 3}$ or on some cycle of $\mathcal{C}$, oriented out of the endpoint. If follows that by a BFS (*breadth-first search*) traversal starting from v_0 in the digraph formed by the above four kinds of oriented

Algorithm A($\overline{G} = (V, \overline{E})$):
Step 1. Compute a maximum 2-matching M;
Step 2. repeatedly find an alternating path to save a singleton in M, till either no singleton exists or no alternating path is found;
Step 3. break cycles in M by removing one edge per cycle, and return the resulting path cover.

Fig. 2 A high-level description of Algorithm A for computing a path cover in the agreement graph $\overline{G} = (V, \overline{E})$

edges, if a vertex on some path of $\mathcal{P}_{\geq 3}$ or on some cycle of $\mathcal{C}$ can be reached then we achieve a simple alternating path; otherwise, we conclude that no alternating path saving the singleton v_0 exists. Both construction of the digraph and the BFS traversal take $O(m)$ time. This proves the lemma.

The second step of the algorithm is to iteratively find a simple alternating path to save a singleton; it terminates when no alternating path is found. The resulting maximum 2-matching is still denoted as M.

In the last step, we break the cycles in M by deleting one edge per cycle to produce a path cover. Denote our algorithm as Algorithm A, of which a high-level description is provided in Fig. 2. We will prove in the next theorem that the path cover produced by Algorithm A contains the minimum number of 0-paths.

Theorem 2.3 Algorithm A *is an* $O(mn \log n)$*-time algorithm for computing a path cover with the minimum number of* 0*-paths in the agreement graph* $\overline{G} = (V, \overline{E})$.

Proof Recall that the last step of Algorithm A is to break cycles only. We thus use the maximum 2-matching achieved at the end of the second step in the following proof, denoted as M. We point out that in the second step, in each iteration where an alternating path is found to save a singleton of the current maximum 2-matching, we swap the edges on the alternating path inside the matching with the edges outside of the matching to *move* from the current maximum 2-matching to another maximum 2-matching which contains one less singleton.

We prove the theorem by the minimal counterexample.

Recall that $\mathcal{P}_0$ is the collection of all singletons (that is, 0-paths) in M. Let M^* be an optimal path cover that contains the minimum number of singletons, and let $\mathcal{P}_0^*$ denote this collection of singletons. Assume to the contrary that the path cover obtained from M contains more than the minimum number of singletons, then we must have

$$|\mathcal{P}_0| > |\mathcal{P}_0^*| \geq 0. \tag{1}$$

Assume that our agreement graph $\overline{G} = (V, \overline{E})$ is a minimal graph on which Eq. (1) holds, then M and M^* should not have any common singleton as otherwise it can be deleted to obtain a smaller graph. That is,

$$\mathcal{P}_0 \cap \mathcal{P}_0^* = \emptyset. \tag{2}$$

It follows that a singleton $v_0 \in \mathcal{P}_0$ is not a singleton in M^*. Suppose $(v_0, v_1) \in M^*$. From the edge maximality of M and the non-existence of an alternating path, we conclude that v_1 has to be the middle vertex of some 2-path $P_1 \in \mathcal{P}_2$.

Let u_1 and v_2 be the two endpoints of the 2-path P_1. For the same reason as for the singleton v_0, from the edge maximality of M and the non-existence of an alternating path, we conclude that in $\overline{G}$ each of u_1 and v_2 can be adjacent to only the middle vertices of 2-paths, including v_1. On the other hand, we conclude that none of u_1 and v_2 can be a singleton in M^*. We prove this by contradiction to assume for instance v_2 is a singleton in M^*; then the alternating path v_0-v_1-v_2 would save v_0 but leave v_2 as a new singleton, which subsequently gives rise to another maximum 2-matching M' with the same number of singletons but M' shares with M^* a common singleton v_2, a contradiction to the minimality of $\overline{G}$.

The last paragraph essentially implies that in M^*, each of u_1 and v_2 is adjacent to the middle vertex of a certain 2-path of $\mathcal{P}_2$. Since the edges (u_1, v_1) and (v_1, v_2) cannot both be in M^* (otherwise v_1 would have degree 3 in M^*), in M^* one of u_1 and v_2, and assume without loss of generality v_2, is adjacent to the middle vertex v_3 of a 2-path $P_2 \in \mathcal{P}_2$ other than P_1.

Let u_2 and v_4 be the two endpoints of the second 2-path P_2. For the same reason as for the singleton v_0, from the edge maximality of M and the non-existence of an alternating path, we conclude that each of u_2 and v_4 can be adjacent to only the middle vertices of 2-paths, including v_1 and v_3. On the other hand, we conclude that none of u_2 and v_4 can be a singleton in M^*, for the same reason as for v_2 in the above. These imply that in M^*, each of u_2 and v_4 is adjacent to the middle vertex of a certain 2-path of $\mathcal{P}_2$. Since there are four endpoints $\{u_1, v_2, u_2, v_4\}$ but only two middle vertices $\{v_1, v_3\}$ for P_1 and P_2, in M^* one of the four endpoints $\{u_1, v_2, u_2, v_4\}$, and assume without loss of generality v_4, is adjacent to the middle vertex v_5 of a 2-path $P_3 \in \mathcal{P}_2$ other than P_1 and P_2.

Let u_3 and v_6 be the two endpoints of the third 2-path P_3. Repeat the same argument as before we have that v_6 is adjacent to the middle vertex v_7 of a fourth 2-path $P_4 \in \mathcal{P}_2$, and so on. These contradict the fact that the graph $\overline{G}$ is finite, and therefore the maximum 2-matching at the end of the second step of Algorithm A contains the minimum number of singletons.

For the running time, since in each iteration of the second step we may "*glue*" all singletons as one for finding an alternating path. If no alternating path is found, then the second step terminates; otherwise one can easily check which singletons are the root of the alternating path and pick to save one of them, and the iteration ends. It follows that there could be $O(n)$ iterations and each iteration needs $O(m)$ time, and thus the total running time for the second step is $O(nm)$. Clearly the last step can be done in $O(n)$ time. Therefore the running time of Algorithm A is dominated by the first step of finding a maximum 2-matching, which is done in $O(mn \log n)$ time. This finishes the proof of the theorem. □

We remark that Gómez and Wakabayashi (2020) independently presented a polynomial time algorithm for computing a path cover without any 0-path, using a similar approach as in our Algorithm A.

2.2 Path cover with the minimum number of {0, 1}-paths

In this variant of the Path Cover problem, given a graph, we aim to find a path cover that contains the minimum total number of 0-paths and 1-paths. Again, the given graph is the complement $\overline{G} = (V, \overline{E})$ of the conflict graph $G = (V, E)$ in $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$. We next present a polynomial time algorithm called Algorithm B that finds for $\overline{G}$ such a path cover.

Recall that in Algorithm A for computing a path cover that contains the minimum number of 0-paths, an alternating path saving a singleton v_0 starts from the singleton v_0 and reaches a vertex v_{2i+1} on a path of $\mathcal{P}_{\geq 3}$ or on a cycle of $\mathcal{C}$ (see Fig. 1). If v_{2i+1} is on a cycle, then the last vertex v_{2i+2} can be any one of the two neighbors of v_{2i+1} on the cycle. If v_{2i+1} is on a k-path, then the last vertex v_{2i+2} is a *non-endpoint* neighbor of v_{2i+1} on the path (the existence is guaranteed by $k \geq 3$); and the reason why v_{2i+2} cannot be an endpoint is obvious since otherwise v_{2i+2} would be left as a new singleton after the edge swapping. In the current variant we want to minimize the total number of 0-paths and 1-paths; clearly v_{2i+2} cannot be an endpoint either and cannot even be the vertex adjacent to an endpoint, for the latter case because the edge swapping saves v_0 but leaves a new 1-path. To guarantee the existence of such vertex v_{2i+2}, the k-path must have $k \geq 4$, and if $k = 4$ then v_{2i+1} cannot be the middle vertex of the 4-path.

Algorithm B is in spirit similar to but in practice slightly more complex than Algorithm A, mostly because the definition of an alternating path saving a singleton or a 1-path is different, and slightly more complex.

In the first step of Algorithm B, we apply the $O(mn \log n)$-time algorithm (Gabow 1983) to find a maximum 2-matching M in $\overline{G}$. Let $\mathcal{P}_0$ ($\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4, \mathcal{P}_{\geq 5}, \mathcal{C}$, respectively) denote the sub-collection of 0-paths (1-paths, 2-paths, 3-paths, 4-paths, paths of length at least 5, cycles, respectively) in M. Also denote $\mathcal{P}_{0,1} = \mathcal{P}_0 \cup \mathcal{P}_1$, $\mathcal{P}_{2,3} = \mathcal{P}_2 \cup \mathcal{P}_3$, and $\mathcal{P}_{2,3,4} = \mathcal{P}_2 \cup \mathcal{P}_3 \cup \mathcal{P}_4$, respectively.

Let $e_0 = (v_0, u_0)$ be an edge in M. In the sequel when we say e_0 is *adjacent* to a vertex v_1 in the graph $\overline{G}$, we mean v_1 is a different vertex (from v_0 and u_0) and at least one of v_0 and u_0 is adjacent to v_1; if both v_0 and u_0 are adjacent to v_1, then pick one (often arbitrarily) for the subsequent purposes. This way, we unify our treatment on singletons and 1-paths, for the reasons to be seen in the following. For ease of presentation, we use an *object* to refer to a vertex or an edge. Like in the last subsection, an ending vertex of a k-path with $k \geq 1$ or an ending edge of a k-path with $k \geq 2$ is called an *end-object* for simplicity.

Let v_0 be a singleton or $e_0 = (v_0, u_0)$ be a 1-path in M. In the underlying graph $\overline{G}$, if v_0 is adjacent to a vertex v_1 on a cycle of $\mathcal{C}$, or on a path of $\mathcal{P}_{\geq 5}$, or on a 4-path such that v_1 is not the middle vertex, then we may delete a certain edge incident at v_1

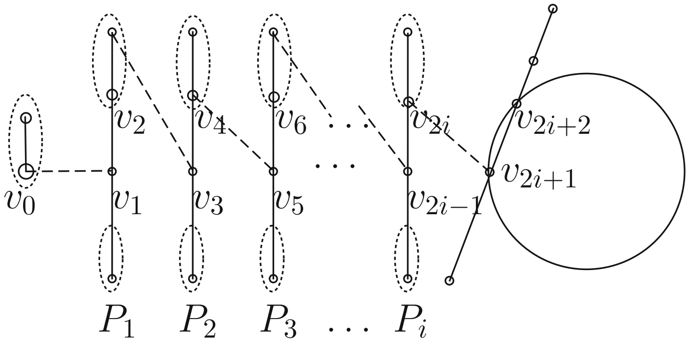

Fig. 3 An alternating path v_0-v_1-v_2-. . .-v_{2i}-v_{2i+1}-v_{2i+2} that saves the singleton v_0 or the 1-path $e_0 = (v_0, u_0)$, where the last two vertices are on a cycle of $\mathcal{C}$, or on a path of $\mathcal{P}_{\geq 5}$, or on a 4-path such that v_{2i+1} is not the middle vertex. In the figure, solid edges are in the maximum 2-matching M, dashed edges are outside of M, and a dotted circle contains an object which is either a vertex or an edge.

from M while add the edge (v_0, v_1) to M to achieve another maximum 2-matching with one less singleton if v_0 is a singleton or with one less 1-path. In either of the three cases, assume the edge deleted from M is (v_1, v_2); then we say the *alternating* path v_0-v_1-v_2 *saves* the singleton v_0 or the 1-path $e_0 = (v_0, u_0)$.

Analogously as in the last subsection, in the general setting, in the underlying graph $\overline{G}$, v_0 is adjacent to a vertex v_1 of a path $P_1 \in \mathcal{P}_{2,3,4}$ (if P_1 is a 4-path then v_1 has to be the middle vertex). Note that this vertex v_1 basically separates the two end-objects of the path P_1 — an analogue to the role of the middle vertex of a 2-path that separates the two endpoints of the 2-path. We say "an end-object of P_1 is adjacent to v_1 via v_2", to mean that if the end-object is a vertex then it is v_2, or if the end-object is an edge, then it is (v_2, u_2), with the edge (v_1, v_2) on the path P_1 (see an illustration in Fig. 3).

Suppose one end-object of P_1, which is adjacent to v_1 via v_2, is adjacent to a vertex v_3 of another $P_2 \in \mathcal{P}_{2,3,4}$ (the same, if P_2 is a 4-path then v_3 has to be the middle vertex); one end-object of P_2, which is adjacent to v_3 via v_4, is adjacent to a vertex v_5 of another $P_3 \in \mathcal{P}_{2,3,4}$ (the same, if P_3 is a 4-path then v_5 has to be the middle vertex); and so on; one end-object of P_{i-1}, which is adjacent to v_{2i-3} via v_{2i-2}, is adjacent to a vertex v_{2i-1} of another $P_i \in \mathcal{P}_{2,3,4}$ (the same, if P_i is a 4-path then v_{2i-1} has to be the middle vertex); one end-object of P_i, which is adjacent to v_{2i-1} via v_{2i}, is adjacent to a vertex v_{2i+1} of a cycle of $\mathcal{C}$, or of a path of $\mathcal{P}_{\geq 5}$, or of a 4-path such that v_{2i+1} is not the middle vertex (see an illustration in Fig. 3), on which a certain edge (v_{2i+1}, v_{2i+2}) is to be deleted. Then we may delete the edges $\{(v_{2j+1}, v_{2j+2}) \mid j = 0, 1, \ldots, i\}$ from M while add the edges $\{(v_{2j}, v_{2j+1}) \mid j = 0, 1, \ldots, i\}$ to M to achieve another maximum 2-matching with one less singleton if v_0 is a singleton or with one less 1-path. We say the *alternating* path v_0-v_1-v_2-. . .-v_{2i}-v_{2i+1}-v_{2i+2} *saves* the singleton v_0 or the 1-path $e_0 = (v_0, u_0)$. It is important to note that in this alternating path, the vertex v_2 "*represents*" the end-object of P_1, meaning that when the end-object is an edge, it is treated very the same as the vertex v_2.

Lemma 2.4 *Given a maximum 2-matching M and an object in $\mathcal{P}_{0,1}$, finding a simple alternating path to save the object, if exists, can be done in $O(m)$ time, where $m = |\overline{E}|$.*

Proof The lemma is a generalization of Lemma 2.2.

Firstly, if an alternating path is not simple, then a cycle that forms a subpath is also alternating and has an even length, and thus the cycle can be removed resulting a shorter alternating path. Repeating this process if necessary, at the end we achieve a simple alternating path. Therefore, we can limit the search for a simple alternating path.

Note that the edges on all possible alternating paths that save an object in $\mathcal{P}_{0,1}$ can be of the following four kinds: 1) all those edges incident at a vertex of the object, each oriented out of the vertex; 2) all those edges of the paths of $\mathcal{P}_{2,3,4}$, each oriented towards an endpoint and each internal edge is bidirected; 3) all those edges each connecting a vertex of an end-object of a path of $\mathcal{P}_{2,3,4}$ to an internal vertex of another such path, oriented from the vertex of the end-object; note that if the second path is a 4-path, then the internal vertex of this 4-path has to be the middle vertex; 4) all those edges each connecting a vertex of an end-object of a path of $\mathcal{P}_{2,3,4}$ to a vertex on some path of $\mathcal{P}_{\geq 5}$, or on some cycle of $\mathcal{C}$, or on some 4-path such that the vertex is not the middle vertex, oriented out of the vertex of the end-object.

If follows that by a BFS traversal starting from the vertices of an object of $\mathcal{P}_{0,1}$ in the digraph formed by the above four kinds of oriented edges, if a vertex on some path of $\mathcal{P}_{\geq 5}$, or on some cycle of $\mathcal{C}$, or on some 4-path such that the vertex is not the middle vertex, can be reached then we achieve a simple alternating path; otherwise, we conclude that no alternating path saving the object exists. Both construction of the digraph and the BFS traversal take $O(m)$ time. This proves the lemma. □

The second step of the algorithm is to iteratively find a simple alternating path to save an object of $\mathcal{P}_{0,1}$; it terminates when no alternating path is found. The resulting maximum 2-matching is still denoted as M.

In the last step, we break the cycles in M by deleting one edge per cycle to produce a path cover. A high-level description of Algorithm B is similar to the one for Algorithm A shown in Fig. 2, replacing a singleton by an object of $\mathcal{P}_{0,1}$. We will prove in Theorem 2.5 that the path cover produced by Algorithm B contains the minimum total number of 0-paths and 1-paths.

Theorem 2.5 Algorithm B *is an $O(mn \log n)$-time algorithm for computing a path cover with the minimum total number of* 0*-paths and* 1*-paths in the agreement graph* $\overline{G} = (V, \overline{E})$.

Proof The proof is similar to the proof of Theorem 2.3.

Recall that the last step of Algorithm B is to break cycles only. We thus use the maximum 2-matching achieved at the end of the second step in the following proof, denoted as M. We point out that in the second step, in each iteration where an alternating path is found to save an object of $\mathcal{P}_{0,1}$ of the current maximum 2-matching, we swap the edges on the alternating path inside in the matching with the edges outside of the matching to *move* from the current maximum 2-matching to another maximum 2-matching (which contains one less object of $\mathcal{P}_{0,1}$).

We prove the theorem by the minimal counterexample.

Let M^* be an optimal path cover that contains the minimum total number of 0-paths and 1-paths, and similarly let $\mathcal{P}_i^*$ denote the sub-collection of the i-paths in M^*, for $i = 0, 1, 2, \ldots$. Assume to the contrary that

$$|\mathcal{P}_{0,1}| > |\mathcal{P}_{0,1}^*| \geq 0, \tag{3}$$

and assume that our agreement graph $\overline{G} = (V, \overline{E})$ is a minimal graph on which Eq. (3) holds, then M and M^* should not have any common singleton or any common 1-path as otherwise it can be deleted to obtain a smaller graph. That is,

$$\mathcal{P}_0 \cap \mathcal{P}_0^* = \emptyset, \text{ and } \mathcal{P}_1 \cap \mathcal{P}_1^* = \emptyset. \tag{4}$$

It follows that an object of $\mathcal{P}_{0,1}$ is not an object of $\mathcal{P}_{0,1}^*$. In the sequel we assume there is a singleton $v_0 \in \mathcal{P}_0$ and show that it leads to a contradiction. A similar contradiction can be constructed if there is a 1-path in $\mathcal{P}_1$. Since v_0 is not a singleton in M^*, we may suppose $(v_0, v_1) \in M^*$. From the edge maximality of M and the non-existence of an alternating path to save v_0, we conclude that v_1 has to be a vertex of some path $P_1 \in \mathcal{P}_{2,3}$ or the middle vertex of some 4-path P_1, such that v_1 separates the two end-objects of P_1.

If an end-object is a vertex, denoted as v_2, then the same from the edge maximality of M and the non-existence of an alternating path to save v_0, we conclude that v_2 behaves the same as v_0, that it can be adjacent to only the vertices of paths in $\mathcal{P}_{2,3}$ or the middle vertices of 4-paths, including v_1. On the other hand, v_2 cannot be a singleton in M^*, since otherwise the alternating path v_0-v_1-v_2 would save v_0 but leave v_2 as a new singleton, which subsequently gives rise to another maximum 2-matching M' with the same number of singletons but M' shares with M^* a common singleton, a contradiction to the minimality of $\overline{G}$.

If an end-object is an edge, denoted as (v_2, u_2), adjacent to v_1 via v_2, then the same from the edge maximality of M and the non-existence of an alternating path to save v_0, we conclude that both v_2 and u_2 behave the same as v_0, that each can be adjacent to only the vertices of paths in $\mathcal{P}_{2,3}$ or the middle vertices of 4-paths, including v_1. On the other hand, at least one of v_2 and u_2 should be adjacent to a third vertex in M^*. Assume this is not the case, then (v_2, u_2) has to be a 1-path in M^* (v_2 and u_2 cannot both be singletons due to the existence of the edge (v_2, u_2)). It follows that the alternating path v_0-v_1-v_2 would save v_0 but leave (v_2, u_2) as a new 1-path, which subsequently gives rise to another maximum 2-matching M' with the same total number of singletons and 1-paths but M' shares with M^* a common 1-path, a contradiction to the minimality of $\overline{G}$.

Since the two path-edges that v_1 is incident to in P_1 cannot both be in M^* (otherwise v_1 would have degree 3 in M^*), in M^* one end-object of the path P_1 is adjacent to a vertex of some path in $\mathcal{P}_{2,3}$ or the middle vertex of some 4-path denoted as P_2. Let v_2 denote the vertex through which this end-object of the path P_1 connects to v_1, and v_3 denote the vertex on the path P_2.

Similarly as in the proof of Theorem 2.3, we may repeat the above argument on the path P_1 for the new path P_2, to either contradict the minimality of the graph $\overline{G}$ or introduce another new path P_3; and so on. The latter cases together contradict the fact that the graph $\overline{G}$ is finite, and therefore the maximum 2-matching at the end of the second step of ALGORITHM B contains the minimum total number of singletons and 1-paths.

For the running time, since in each iteration of the second step again we may "*glue*" all singletons and the endpoints of all the 1-paths as one for finding an alternating path. If no alternating path is found, then the second step terminates; otherwise one can easily check which singletons and/or 1-paths are the root of the alternating path and pick to save one of them, and the iteration ends. It follows that there could be $O(n)$ iterations and each iteration needs $O(m)$ time, and thus the total running time for the second step is $O(nm)$. Clearly the last step can be done in $O(n)$ time. Therefore the running time of ALGORITHM B is dominated by the first step of finding a maximum 2-matching, which is done in $O(mn \log n)$ time. This finishes the proof of the theorem.

Remark 2.6 The path cover produced by ALGORITHM B has the minimum total number of 0-paths and 1-paths in the agreement graph $\overline{G} = (V, \overline{E})$. One may run ALGORITHM A at the end of the second step of ALGORITHM B to achieve a path cover with the minimum total number of 0-paths and 1-paths, and with the minimum number of 0-paths. During the execution of ALGORITHM A, a singleton trades for a 1-path.

2.3 Approximation algorithms for $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$

Given an instance of the problem $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$, where there are n unit jobs $V = \{J_1, J_2, \ldots, J_n\}$ to be processed on the two-machine flow-shop, with their conflict graph $G = (V, E)$, we want to find a schedule with a provable makespan.

For a k-path in the agreement graph $\overline{G} = (V, \overline{E})$, where $k \ge 0$, for example $P = J_1$-J_2-…-J_k-J_{k+1}, we compose a sub-schedule π_P in which the machine M_1 continuously processes the jobs $J_1, J_2, \ldots, J_{k+1}$ in order, and the machine M_2 in one unit of time after M_1 continuously processes these jobs in the same order. The sub-makespan for the flow-shop to complete these $k+1$ jobs is thus $k+2$ (units of time). Let $M = \{P_1, P_2, \ldots, P_\ell\}$ be a path cover of size ℓ in the agreement graph $\overline{G}$. For each path P_i we use $|P_i|$ to denote its length and construct the sub-schedule π_{P_i} as above that has a sub-makespan of $|P_i| + 2$. We then concatenate these ℓ sub-schedules (in an arbitrary order) into a full schedule π, which clearly has a makespan

$$C^{\pi}_{\max} = \sum_{i=1}^{\ell} (|P_i| + 2) = n + \ell. \tag{5}$$

On the other hand, given a schedule π, if two jobs J_{j_1} and J_{j_2} are processed concurrently on the two machines, then they have to be agreeing to each other and thus adjacent in the agreement graph $\overline{G}$; we select this edge (J_{j_1}, J_{j_2}). Note that one job can be processed concurrently with at most two other jobs as there are only two machines. Therefore, all the selected edges form into a number of vertex-disjoint paths in $\overline{G}$ (due to the flow-shop, no cycle is formed); these paths together with the vertices outside of the paths, which are the 0-paths, form a path cover for $\overline{G}$. Assuming without loss of generality that two machines cannot both idle at any time point, the makespan of the schedule is exactly calculated as in Eq. (5).

We state this relationship between a feasible schedule and a path cover in the agreement graph $\overline{G}$ into the following lemma.

Lemma 2.7 Tellache and Boudhar (2018) *A feasible schedule π for the problem $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ one-to-one corresponds to a path cover M in the agreement graph $\overline{G}$, and $C^{\pi}_{\max} = n + |M|$, where n is the number of jobs in the instance.*

Theorem 2.8 *The problem $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ admits an $O(mn \log n)$-time $4/3$-approximation algorithm, where $n = |V|$ and $m = |\overline{E}|$.*

Proof Let π^* denote an optimal schedule for the problem $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ with a makespan $C^*_{\max}$, and M^* be the corresponding path cover in the agreement graph $\overline{G}$. The sub-collection of 0-paths and 1-paths in M^* is denoted as $\mathcal{P}^*_{0,1}$.

Let M be the path cover computed by Algorithm B for the agreement graph $\overline{G}$ that achieves the minimum total number of 0-paths and 1-paths. The sub-collections of 0-paths and 1-paths in M are denoted as $\mathcal{P}_0$ and $\mathcal{P}_1$, respectively, and $\mathcal{P}_{0,1}$ denotes their union. Then we have

$$|\mathcal{P}^*_{0,1}| \geq |\mathcal{P}_{0,1}|.$$

From Lemma 2.7, we have

$$C^*_{\max} = n + |M^*| \geq n + |\mathcal{P}^*_{0,1}| \geq n + |\mathcal{P}_{0,1}|.$$

It follows also from Lemma 2.7 that the schedule constructed using the path cover M has a makespan

$$C_{\max} = n + |M| \leq n + |\mathcal{P}_{0,1}| + \frac{1}{3}\left(n - |\mathcal{P}_0| - 2|\mathcal{P}_1|\right) \leq \frac{4}{3} C^*_{\max},$$

since every path of length 2 or above contains at least three vertices.

Note that the running time of the approximation algorithm is obvious, which calls Algorithm B and then constructs the schedule in $O(n)$ time using the computed path cover.

Remark 2.9 If Algorithm A is used in the proof of Theorem 2.8 to compute a path cover with the minimum number of 0-paths and subsequently to construct a schedule π, then we have $C^{\pi}_{\max} \leq \frac{3}{2} C^*_{\max}$. That is, we have an $O(mn \log n)$-time $3/2$-approximation algorithm based on Algorithm A.

When the agreement graph $\overline{G}$ consists of k vertex-disjoint triangles such that a vertex of the i-th triangle is adjacent to a vertex of the $(i+1)$-st triangle, for $i = 1, 2, \ldots, k-1$, and the maximum degree is 3, Algorithm B could produce a path cover containing k 2-paths, while there is a Hamiltonian path in the graph. This suggests that the approximation ratio $4/3$ is asymptotically tight.

3 Approximating $F2 \mid G = K_\ell \cup K_{n-\ell}, p_{ij} \mid C_{\max}$

In this section, we present a 3/2-approximation algorithm for the weakly NP-hard problem $F2 \mid G = K_\ell \cup K_{n-\ell}, p_{ij} \mid C_{\max}$ for arbitrary jobs with a conflict graph that is the union of two disjoint cliques. Clearly, the agreement graph $\overline{G} = K_{\ell,n-\ell}$ is a complete bipartite graph. Without loss of generality, let the job set of K_ℓ be $A = \{J_1, J_2, \dots, J_\ell\}$ and the job set of $K_{n-\ell}$ be $B = \{J_{\ell+1}, J_{\ell+2}, \dots, J_n\}$.

For the job set A, we merge all its jobs (in the sequential order with increasing indices) to become a single "*aggregated*" job denoted as J_A, with its processing time on the machine M_1 being $P_A^1 = \sum_{j=1}^{\ell} p_{1j}$ and its processing time on the machine M_2 being $P_A^2 = \sum_{j=1}^{\ell} p_{2j}$. Likewise, for the job set B, we merge all its jobs (in the sequential order with increasing indices) to become a single aggregated job denoted as J_B, with its two processing times being $P_B^1 = \sum_{j=\ell+1}^{n} p_{1j}$ and $P_B^2 = \sum_{j=\ell+1}^{n} p_{2j}$, respectively. We now have an instance of the classical two-machine flow-shop scheduling problem consisting of only two aggregated jobs J_A and J_B, and we may apply Johnson's algorithm (Johnson 1954) to determine the processing order for J_A and J_B — a schedule denoted as π. From π we obtain a schedule for the original instance, by expanding the aggregated jobs J_A and J_B back, of the problem $F2 \mid G = K_\ell \cup K_{n-\ell}, p_{ij} \mid C_{\max}$, which is also denoted as π since there is no major difference. We call this algorithm as Algorithm C.

Theorem 3.1 Algorithm C *is an* $O(m)$*-time* 3/2*-approximation algorithm for the problem* $F2 \mid G = K_\ell \cup K_{n-\ell}, p_{ij} \mid C_{\max}$*, where* m *is the number of edges in the conflict graph* G.

Proof Firstly, we note that Algorithm C needs to spend $O(m)$ time to recognize that the conflict graph is indeed the union of two disjoint cliques, and subsequently composes the two aggregated jobs. If the two job subsets A and B are given without the need of recognition, then composing the two aggregated jobs can be done in $O(n)$ time, where n is the number of given jobs. Scheduling two jobs on the two-machine flow-shop, that is, by Johnson's algorithm, is done in constant time, afterwards the schedule π for the original n jobs can be constructed in $O(n)$ time.

Let $C^*_{\max}$ and $C^\pi_{\max}$ denote the optimal makespan and the makespan of the schedule π produced by Algorithm C, respectively. One clearly sees that

$$C^*_{\max} \ge \max\{P_A^1 + P_A^2, P_A^1 + P_B^1, P_B^1 + P_B^2, P_A^2 + P_B^2\}, \tag{6}$$

in which the four sums represent the total processing time of jobs in A, the total processing time of jobs on the machine M_1, the total processing time of jobs in B, and the total processing time of jobs on the machine M_2, respectively.

Assume without loss of generality that $P_A^1 \le P_B^1$:

- If $P_A^1 \le P_A^2$, then $C^\pi_{\max} = P_A^1 + \max\{P_A^2, P_B^1\} + P_B^2 \le P_A^1 + C^*_{\max} \le \frac{3}{2} C^*_{\max}$;
- if $P_A^1 > P_A^2 > P_B^2$, then $C^\pi_{\max} = P_A^1 + \max\{P_A^2, P_B^1\} + P_B^2 \le C^*_{\max} + P_B^2 \le \frac{3}{2} C^*_{\max}$;
- if $P_A^1 > P_A^2$ and $P_A^2 \le P_B^2$, then $C^\pi_{\max} = P_B^1 + \max\{P_B^2, P_A^1\} + P_A^2 \le C^*_{\max} + P_A^2 \le \frac{3}{2} C^*_{\max}$.

This proves the theorem. □

In the schedule produced by Algorithm C, one sees that when the jobs of A are processed on the machine M_1, the other machine M_2 is left idle. This is certainly disadvantageous. For instance, when the jobs are all unit jobs and $|A| = |B| = \frac{1}{2}n$, the makespan of the produced schedule is $\frac{3}{2}n$, while the agreement graph is Hamiltonian and thus by Eq. (5) the optimal makespan is only $n+1$. This huge gap suggests that one could probably design a better approximation algorithm and we leave it as an open question.

4 Concluding remarks

In this paper, we investigated the approximation algorithms for the two-machine flow-shop scheduling problem with a conflict graph, in particular a special case of all unit jobs and another special case where the conflict graph is the union of two disjoint cliques, that is, $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ and $F2 \mid G = K_\ell \cup K_{n-\ell}, p_{ij} \mid C_{\max}$, respectively. For the first problem we studied the graph theoretical problem of finding a path cover with the minimum total number of 0-paths and 1-paths, and presented a polynomial time exact algorithm. This exact algorithm leads to a 4/3-approximation algorithm for the problem $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$. We also showed that the performance ratio 4/3 is asymptotically tight. For the second problem $F2 \mid G = K_\ell \cup K_{n-\ell}, p_{ij} \mid C_{\max}$, we presented a 3/2-approximation algorithm.

We conjecture that designing approximation algorithms for $F2 \mid G = (V, E), p_{ij} = 1 \mid C_{\max}$ with a performance ratio better than 4/3 is challenging, since one way or the other one has to deal with long paths in a path cover or has to deal with the original Path Cover problem. We in fact strongly suspect that the problem is APX-hard. Nevertheless, better approximation algorithms for $F2 \mid G = K_\ell \cup K_{n-\ell}, p_{ij} \mid C_{\max}$ can be expected.

Acknowledgements Y. Chen, G. Chen, and A. Zhang are supported by the NSFC Grants 11771114, 11971139 and the Zhejiang Provincial NSFC Grant LY21A010014; Y. Chen and A. Zhang are also supported by the China Scholarship Council Grants 201508330054 and 201908330090, respectively. R. Goebel, G. Lin, and L. Liu are supported by NSERC Canada; L. Liu is also supported by the Fundamental Research Funds for the Central Universities (Grant No. 20720160035) and the China Scholarship Council Grant No. 201706315073.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Asdre K, Nikolopoulos SD (2007) A linear-time algorithm for the k-fixed-endpoint path cover problem on cographs. Networks 50:231–240

Asdre K, Nikolopoulos SD (2010) A polynomial solution to the k-fixed-endpoint path cover problem on proper interval graphs. Theor Comput Sci 411:967–975
Baker BS, Coffman EG (1996) Mutual exclusion scheduling. Theor Comput Sci 162:225–243
Bendraouche M, Boudhar M (2012) Scheduling jobs on identical machines with agreement graph. Comput Oper Res 39:382–390
Bendraouche M, Boudhar M (2016) Scheduling with agreements: new results. Int J Prod Res 54:3508–3522
Błażewicz J, Kubiak W, Szwarcfiter J (1988) Scheduling unit - time tasks on flow - shops under resource constraints. Ann Oper Res 16:255–266
Blazewicz J, Lenstra JK, Rinnooy Kan AHG (1983) Scheduling subject to resource constraints: classification and complexity. Discret Appl Math 5:11–24
Bodlaender HL, Jansen K (1995) Restrictions of graph partition problems. Part I. Theor Comput Sci 148:93–109
Cai Y, Chen G, Chen Y, Goebel R, Lin G, Liu L, Zhang A (2018) Approximation algorithms for two-machine flow-shop scheduling with a conflict graph. In: Proceedings of the 24th international computing and combinatorics conference (COCOON 2018), LNCS 10976, pp 205–217
Chen B, Glass CA, Potts CN, Strusevich VA (1996) A new heuristic for three-machine flow shop scheduling. Oper Res 44:891–898
Even G, Halldórsson MM, Kaplan L, Ron D (2009) Scheduling with conflicts: online and offline algorithms. J Schedul 12:199–224
Gabow HN (1983) An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In: Proceedings of the 15th annual ACM symposium on theory of computing (STOC'83), pp 448–456
Garey MR, Graham RL (1975) Bounds for multiprocessor scheduling with resource constraints. SIAM J Comput 4:187–200
Garey MR, Johnson DS (1975) Complexity results for multiprocessor scheduling under resource constraints. SIAM J Comput 4:397–411
Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman and Company, San Francisco
Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and jobshop scheduling. Math Oper Res 1:117–129
Garey MR, Johnson DS, Tarjan RE (1976) The planar Hamiltonian circuit problem is NP-complete. SIAM J Comput 5:704–714
Golumbic MC (2004) Algorithmic graph theory and perfect graphs. Elsevier, Amsterdam
Gómez R, Wakabayashi Y (2020) Nontrivial path covers of graphs: existence, minimization and maximization. J Combin Optim 39:437–456
Gonzalez T, Sahni S (1978) Flowshop and jobshop schedules: complexity and approximation. Oper Res 26:36–52
Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann Discret Math 5:287–326
Hall LA (1998) Approximability of flow shop scheduling. Math Program 82:175–190
Halldórsson MM, Kortsarz G, Proskurowski A, Salman R, Shachnai H, Telle JA (2003) Multicoloring trees. Inf Comput 180:113–129
Johnson SM (1954) Optimal two- and three-machine production schedules with setup times included. Naval Res Logist 1:61–68
Müller H (1996) Hamiltonian circuits in chordal bipartite graphs. Discret Math 156:291–298
Pao LL, Hong CH (2008) The two-equal-disjoint path cover problem of matching composition network. Inf Process Lett 107:18–23
Rizzi R, Tomescu AI, Mäkinen V (2014) On the complexity of minimum path cover with subpath constraints for multi-assembly. BMC Bioinform 15:S5
Röck H (1983) Scheduling unit task shops with resource constraints and excess usage costs. Technical Report, Fachbereich Informatik, Technical University of Berlin, Berlin
Röck H (1984) Some new results in flow shop scheduling. Zeitschrift für Oper Res 28:1–16
Süral H, Kondakci S, Erkip N (1992) Scheduling unit-time tasks in renewable resource constrained flowshops. Zeitschrift für Oper Res 36:497–516
Tellache NEH, Boudhar M (2017) Two-machine flow shop problem with unit-time operations and conflict graph. Int J Prod Res 55:1664–1679

Tellache NEH, Boudhar M (2018) Flow shop scheduling problem with conflict graphs. Ann Oper Res 261:339–363

Williamson DP, Hall LA, Hoogeveen JA, Hurkens CAJ, Lenstra JK, Sevastianov SV, Shmoys DB (1997) Short shop schedules. Oper Res 45:288–294

Affiliations

Yong Chen[1] · Yinhui Cai[1] · Longcheng Liu[2] · Guangting Chen[3] · Randy Goebel[4] · Guohui Lin[4] · Bing Su[5] · An Zhang[1]

✉ Guohui Lin
guohui@ualberta.ca

✉ Bing Su
subing684@sohu.com

Yong Chen
chenyong@hdu.edu.cn

Yinhui Cai
1072785809@qq.com

Longcheng Liu
longchengliu@xmu.edu.cn

Guangting Chen
gtchen@hdu.edu.cn

Randy Goebel
rgoebel@ualberta.ca

An Zhang
anzhang@hdu.edu.cn

1 Department of Mathematics, Hangzhou Dianzi University, Hangzhou, China

2 School of Mathematical Sciences, Xiamen University, Xiamen, China

3 Taizhou University, Taizhou, China

4 Department of Computing Science, University of Alberta, Edmonton, AB T6G 2E8, Canada

5 School of Economics and Management, Xi'an Technological University, Xi'an, China