3
A Faster Algorithm for Two Machine Flow Shop Scheduling

3.1 Introduction

This chapter explores the two machine flow shop scheduling problem (2-PFS), a particular case of the PFS problem in which the jobs must be processed in exactly two machines. As previously stated, the 2-PFS problem can be solved in polynomial time due to the classical algorithm of Johnson [34]. In fact, during the last fifty years this $\Theta(n \log n)$ algorithm has been considered the most efficient method for solving the 2-PFS problem[49].

The main contribution of this work is to introduce a new and faster algorithm for the 2-PFS problem. This algorithms runs in $\Theta(n \log \kappa)$ time , where n is the number of jobs and κ is the minimum number of cliques necessary to cover a specific interval graph. This interval graph $G = (V, E)$ is obtained from the 2-PFS instance in $\Theta(n)$ time. In particular, $|V| = n$. Since that, by the definition of a clique covering, $\kappa \leq |V|$, this new algorithm is asymptotically faster than the previous from Johnson. From the best of our knowledge, this is the current faster algorithm that solves the 2-PFS problem.

It is important to stress that this new algorithm is not based on faster *sorting* methods. Our algorithm may use any $\Theta(n \log n)$ comparison based sorting method to achieve the claimed $\Theta(n \log \kappa)$ time complexity. In fact, our results are based on the fact that it is sufficient to partition the jobs into κ disjoint subsets in $\Theta(n \log \kappa)$ time and order κ elements in $\Theta(\kappa \log \kappa)$ time to achieve an optimal solution. Theoretical results supporting this new algorithm provide a more precise characterization of optimal permutation schedules for 2-PFS instances. Finally, the connection established by this new algorithm between 2-PFS and the minimum clique covering problem is, to the best of our knowledge, completely new.

This chapter is organized as follows. At section 3.2 we review the classical algorithm of Johnson [34], proving its correctness and time complexity analysis. Section 3.3 explores the property of job set splitting, which holds for any 2-PFS instance. Section 3.4 introduces the concept of job cluster, a subset of jobs for which optimal permutation schedules can be determined in linear time. Section 3.5 extends the study of optimal permutation schedules to job sets consisting of two or more clusters. Section 3.6 reviews the definition of interval graph. Section 3.7 introduces the concepts of cliques and covers of a graph. Section 3.8 explores the connection between the 2-PFS problem and clique covering on interval graphs. Final conclusions are considered at section 3.9.

3.2 Johnson's Algorithm

First of all, let us assume that the **concatenation** of two permutations $\pi = \langle a_{\pi_1}, \ldots, a_{\pi_n} \rangle$ and $\varphi = \langle b_{\varphi_1}, \ldots, b_{\varphi_m} \rangle$ is defined as: $\pi \circ \varphi = \langle a_{\pi_1}, \ldots, a_{\pi_n}, b_{\varphi_1}, \ldots, b_{\varphi_m} \rangle$.

Algorithm 2: Johnson's Algorithm

Input : Set $\mathbb{J}$ of n jobs,
Processing times matrix $T \in \Re^{+}_{2\times\mathbb{J}}$.
Output: permutation schedule $\pi : \{1, \ldots, n\} \mapsto \mathbb{J}$.
begin
  $A \longleftarrow \emptyset$; $B \longleftarrow \emptyset$;
  for *each job* $j \in \mathbb{J}$ **do**
    if $t_{1,j} \leq t_{2,j}$ **then**
      $A \longleftarrow A \cup \{j\}$;
    else
      $B \longleftarrow B \cup \{j\}$;
  $\pi_A \longleftarrow$ Sorting of A in non-decreasing order of $t_{1,j}$;
  $\pi_B \longleftarrow$ Sorting of B in non-increasing order of $t_{2,j}$;
  $\pi \longleftarrow \pi_A \circ \pi_B$;
  return π ;
end

Theorem 20 *Johnson's Algorithm generates optimal permutation schedules for the 2-PFS problem and can be implemented in* $\Theta(n \log n)$ *time.*

Proof: Suppose, by contradiction, that the permutation $\pi^\star$ generated by Johnson'a algorithm is not optimal, i.e., there exists a permutation $\pi \neq \pi^\star$ such that $makespan(\pi) \leq makespan(\pi^\star)$. Once $\pi \neq \pi^\star$, there is a pair of jobs j and k adjacent on π, i.e., $\pi^{-1}(k) = \pi^{-1}(j) + 1$, satisfying one of the following conditions:

1. $j \in B$ and $k \in A$.

2. $j \in A$, $k \in A$ and $t_{1,j} > t_{1,k}$.

3. $j \in B$, $k \in B$ and $t_{2,j} < t_{2,k}$.

It will be proved that switching the positions of jobs j and k on permutation π, the makespan is not increased. More precisely, let $C(1,k)$ and $C(2,k)$ denote the completion time of job k on first and second machines, respectively, under the schedule π. Similarly, let $C'(1,j)$ and $C'(2,j)$ denote such completion times of job j after switching positions. It will be demonstrated that since $C'(1,j) \leq C(1,k)$ and $C'(2,j) \leq C(2,k)$, the pairwise interchange does not increase the makespan.

Let us prove first that $C'(1,j) \leq C(1,k)$. Due to the makespan recurrence, the completion time of job k on the first machine, under schedule π, is:

$$C(1,k) = C(1,i) + t_{1,j} + t_{1,k} \tag{1}$$

Switching the positions of jobs j and k, the completion time of job j on the first machine would be:

$$C'(1,j) = C'(1,i) + t_{1,k} + t_{1,j} \tag{2}$$

Once $C'(1,i) = C(1,i)$ follows that $C'(1,j) = C(1,k)$.

Let us prove now that $C'(2,j) \leq C(2,k)$. In order to prove this statement, consider the existence of a job i prior to j on π. Following the makespan recurrence, the completion time of job k on the second machine, under schedule π, is:

$$C(2,k) = \max\{C(1,k), C(2,j)\} + t_{2,k} \tag{3}$$

Once $C(1,k) = C(1,j) + t_{1,k}$ and $C(2,j) = \max\{C(1,j), C(2,i)\} + t_{2,j}$ equation 3 can be rewritten as:

$$C(2,k) = \max\{C(1,j) + t_{1,k}, C(1,j) + t_{2,j}, C(2,i) + t_{2,j}\} + t_{2,k} \tag{4}$$

Since $C(1,j) = C(1,i) + t_{1,j}$, we have:

$$C(2,k) = \max \begin{cases} C(1,i) + t_{1,j} + t_{1,k} + t_{2,k} \\ C(1,i) + t_{1,j} + t_{2,j} + t_{2,k} \\ C(2,i) + t_{2,j} + t_{2,k} \end{cases} \tag{5}$$

Let us consider now the effect, on the second machine, of a pairwise interchange of jobs j and k. Once $C'(1,i) = C(1,i)$ and $C'(2,i) = C(2,i)$ equation 5 changes to:

$$C'(2,j) = \max \begin{cases} C(1,i) + t_{1,j} + t_{1,k} + t_{2,j} \\ C(1,i) + t_{1,k} + t_{2,j} + t_{2,k} \\ C(2,i) + t_{2,j} + t_{2,k} \end{cases} \quad (6)$$

As mentioned before, three cases must be considered now.

Case I: $j \in B$ and $k \in A$. So, $t_{1,j} > t_{2,j}$ and $t_{1,k} \leq t_{2,k}$. As consequence, the first term of equation 6 is not greater than the second term of equation 5, the second term of equation 6 is not greater than the first term of equation 5 and the third term of both equations are identical. Therefore, $C'(2,j) \leq C(2,k)$.

Case II: $j \in A$, $k \in A$ and $t_{1,j} > t_{1,k}$. So, $t_{1,j} \leq t_{2,j}$ and $t_{1,k} \leq t_{2,k}$. As consequence, the first and second terms of equation 6 are not greater than the second term of equation 5 and the third term of both equations are identical. Therefore, $C'(2,j) \leq C(2,k)$

Case III: $j \in B$, $k \in B$ and $t_{2,j} < t_{2,k}$. So, $t_{2,j} < t_{1,j}$ and $t_{2,k} < t_{2,j}$. As consequence, the first and second terms of equation 6 are not greater than the first term of equation 5 and the third term of both equations are identical. Therefore, $C'(2,j) \leq C(2,k)$

Time complexity of Johnson's algorithm is dominated by the sorting phase of sets A and B. It can be implemented in $\Theta(n \log n)$ by using any optimal comparison based sorting method like *mergesort*, *heapsort* or *quicksort*. The remaining steps of the algorithm can be executed in linear time. ■

3.3 Splitting the Job Set

In this section, we explore an important property of the `2-PFS` problem which will be used during this chapter. Due to this property, it is possible to obtain an optimal solution for a `2-PFS` instance $\mathbb{J}$ by its decomposition into two `2-PFS` instances A and B, $A \bigcup B = \mathbb{J}$, $A \bigcap B = \emptyset$, where A contains all jobs with its maximum processing times on the first machine and B gets all remaining jobs. In particular, one can demonstrate that an optimal permutation schedule for $\mathbb{J}$ can be obtained by a simple concatenation of the optimal permutations for A and B. The following corollary of theorem 20 formally states this result.

Corollary 21 *Let $\mathbb{J}$ be a set of n jobs from a* `2-PFS` *instance, $A \subseteq \mathbb{J}$ be the subset of jobs whose processing times on the first machine are not greater than its processing times on the second machine and $B = \mathbb{J} \backslash A$. Let π_A and π_B be optimal permutation schedules for the* `2-PFS` *instances A and B, respectively. Then, $\pi^\star = \pi_A \circ \pi_B$ is an optimal permutation schedule for $\mathbb{J}$.*

Proof: Let $|A| = a$, $|B| = b$ and π denote the permutation returned from Johnson's algorithm by taking $\mathbb{J}$ as input. Let π_A and π_B be permutations over sets A and B such that $\pi = \pi_A \circ \pi_B$.

From makespan definition, the completion time of job $\pi(k)$ on the first machine is $C(1, \pi(k)) = \sum_{i=1}^{k} t_{1,\pi(i)}$. The completion time of job $\pi(k)$ on the second machine is given by the recurrence equation:

$$C(2, \pi(k)) = \max\{C(1, \pi(k)), C(2, \pi(k-1))\} + t_{2,\pi(k)} \tag{7}$$

The basis of the recurrence is $k = 1$, when $C(2, \pi(k)) = t_{1,\pi(1)} + t_{2,\pi(1)}$.

Equation 7 lead us to deduce that there will always exist one index $k' \in \{1, \ldots, n\}$ such that $C(2, \pi(k'))$ is a recurrence invoked by the calculation of $makespan(\pi) = C(2, \pi(n))$ and $C(2, \pi(k')) = C(1, \pi(k')) + t_{2,\pi(k')}$. This proof follows by contradiction. Suppose such k' does not exist. Therefore, every recurrence on the form $C(2, \pi(k'))$ invoked by the calculation of $makespan(\pi)$ is given by $C(2, \pi(k')) = C(2, \pi(k'-1)) + t_{2,\pi(k')}$. However, by definition this property does not hold on the basis of equation 7, proving the assertion.

The existence of such k' implies that $makespan(\pi)$ can be rewritten as:

$$makespan(\pi) = C(2, \pi(n)) = \sum_{i=1}^{k'} t_{1,\pi(i)} + \sum_{j=k'}^{n} t_{2,\pi(j)} \tag{8}$$

Depending on the value of k', two cases must be considered:

Case (I): $k' \in \{1, \ldots, a\}$. In this case, all jobs of set B contributes to the sum 8 with its processing times on the second machine. Set A contributes to the sum 8 with the completion time of its last job, on the second machine, following permutation π_A. Hence, $C(2, \pi(n)) = \sum_{j \in B} t_{1,j} + C_A(2, \pi_A(a))$.

Case (II): $k' \in \{a+1, \ldots, n\}$. In this case, all jobs of set A contributes to the sum 8 with its processing times on the first machine. Set B contributes to the sum 8 with the completion time of its last job, on the second machine, following permutation π_B. Hence, $C(2, \pi(n)) = \sum_{j \in A} t_{1,j} + C_B(2, \pi_B(b))$.

Therefore, the makespan resulting from permutation π becomes:

$$makespan(\pi) = \max \begin{cases} \sum_{j \in A} t_{1,j} + C_B(2, \pi_B(b)) \\ \sum_{j \in B} t_{2,j} + C_A(2, \pi_A(a)) \end{cases} \quad (9)$$

It is important to stress that, in order to reach equation 9, it was never used the fact that π is optimal for $\mathbb{J}$. In particular, the only property of π leading to equation 9 was that π is a concatenation of two permutations π_A and π_B over sets A and B, respectively.

Let us consider now the effect of applying permutation $\pi^\star = \pi^\star_A \circ \pi^\star_B$ over $\mathbb{J}$. Once equation 9 is valid for $\pi^\star$, makespan($\pi^\star$) can be calculated as:

$$makespan(\pi^\star) = \max \begin{cases} \sum_{j \in A} t_{1,j} + C_B(2, \pi^\star_B(b)) \\ \sum_{j \in B} t_{2,j} + C_A(2, \pi^\star_A(a)) \end{cases} \quad (10)$$

Once $\pi^\star_A$ and $\pi^\star_B$ are optimal, $C_A(2, \pi^\star_A(a)) \leq C_A(2, \pi^\star_A(a))$ and $C_B(2, \pi^\star_B(b)) \leq C_B(2, \pi^\star_B(b))$ follows that $makespan(\pi^\star) \leq makespan(\pi)$. Since π is, by definition, an optimal permutation, $\pi^\star$ is either. ■

Corollary 21 proves that the decomposition of the original job set $\mathbb{J}$ into two disjoint sets A and B followed by the concatenation of the optimal permutation π_A and π_B, for A and B, respectively, is a *sufficient* condition to obtaining an optimal permutation schedule for $\mathbb{J}$. However, this is not a *necessary* condition. In fact, there exist `2-PFS` instances admitting optimal solutions in which the jobs of set A not necessarily come before the jobs of set B.

Another point to be considered is that the partial solutions π_A and π_B returned by Johnson's algorithm are always optimal for the instances A and B, respectively. To see that this is true for set A , consider a `2-PFS` instance in which every job has its maximum processing time on the second machine. Therefore, $A = \mathbb{J}$ and $B = \emptyset$. Once that by theorem 20 Johnson's algorithm is optimal for any `2-PFS` instance, it is optimal for A in particular. Similar arguments leads to the optimality of π_B.

However, as demonstrated by corollary 21, every solution for sets A and B that minimizes the maximum of the expression 9 is optimal. In other words, if $\sum_{j \in A} t_{1,j} + C_B(2, \pi_B(b)) > \sum_{j \in B} t_{2,j} + C_A(2, \pi_A(a))$ we need an optimal permutation for B, but any permutation for A is valid. Hence, there exist at least $|A|!$ optimal solutions for $\mathbb{J}$. Otherwise, an optimal permutation for A is necessary,

the permutation for set B is irrelevant and there are at least $|B|!$ optimal solutions for $\mathbb{J}$.

Finally, it is an open problem to obtain a complete characterization of the class of optimal solutions for the 2-PFS problem. Pinedo [49] said that "*Johnson's schedules are by no means the only schedules that are optimal for 2-PFS. The class of optimal schedules appears to be hard to characterize and data dependent*"' . One of the contributions of this work, in particular, is to provide a better understanding on this class.

Lemma 22 *Let B be a set of n jobs such that $t_{1,j} \geq t_{2,j}$ for every $j \in B$. Let A be another set of jobs constructed from B as follows: for every job $j \in B$ create a job $j' \in A$ such that $t_{1,j'} = t_{2,j}$ and $t_{2,j'} = t_{1,j}$. Let π be an optimal permutation for the 2-PFS instance A. Then, the reverse of permutation π, denoted by $\sigma = \overline{\pi}$, is an optimal solution for the 2-PFS instance B.*

Proof: Equation 8, found in the proof of corollary 21, states that the makespan resulting from a permutation schedule of a 2-PFS instance can be calculated by the addition of two distinct summations. As consequence, for the job set A and permutation schedule π there is an index $k \in \{1, \ldots, n\}$ that maximizes the following expression:

$$makespan(\pi) = C_A(2, \pi(n)) = \sum_{i=1}^{k} t_{1,\pi(i)} + \sum_{j=k}^{n} t_{2,\pi(j)} \tag{11}$$

Due to the existing relation between sets A and B, for every job $\pi(i) \in B$ there is a job $\pi(i)' \in A$ such that $t_{1,\pi(i)'} = t_{2,\pi(i)}$ and $t_{2,\pi(i)'} = t_{1,\pi(i)}$. Therefore,

$$makespan(\pi) = \sum_{i=1}^{k} t_{2,\pi(i)'} + \sum_{j=k}^{n} t_{1,\pi(j)'}. \tag{12}$$

By the definition of reverse permutation, it follows:

$$makespan(\pi) = \sum_{j=1}^{n-k+1} t_{1,\sigma(j)'} + \sum_{i=n-k+1}^{n} t_{2,\sigma(i)'} \tag{13}$$

Let us study now the makespan of permutation $\sigma : \{1, \ldots, n\} \mapsto B$.
Due to equation 8 there is an index $k' \in \{1, \ldots, n\}$ such that $makespan(\sigma)$ can be rewritten as:

$$makespan(\sigma) = C_B(2, \sigma(n)) = \sum_{j=1}^{k'} t_{1,\sigma(j)'} + \sum_{i=k'}^{n} t_{2,\sigma(i)'} \tag{14}$$

Since the indexes $n - k + 1$ and k' are selected in order to maximize identical expressions on the right-hand of equations 13 and 14 we have that $makespan(\pi) = makespan(\sigma)$. ■

Lemma 22 is a valuable tool on the design of algorithms for the 2-PFS problem. It claims that it is possible to reduce the problem of determining an optimal permutation schedule for an instance A, formed by jobs with maximum processing times on the second machine, to the problem of finding an optimal permutation for an equivalent instance B of jobs with maximum processing times on the first machine. This reduction is also valid on the opposite direction, from B to A. Furthermore, as stated by lemma 22, it can be done in linear time on the number of jobs.

In this sense, from this point to the end of the chapter, lemma 22 will be used in conjunction with corollary 21. As consequence, every 2-PFS instance $\mathbb{J}$ will be considered previously partitioned into two disjoint instances A and B. In particular, our focus will be on finding optimal solutions for the instance A, with maximum processing times on the second machine. It must be clear, by lemma 22, that the same algorithm used for A can be used for B. Finally, as stated by corollary 21, from a simple concatenation of the optimal solutions π_A and π_B for A and B, respectively, it is possible to determine an optimal solution for $\mathbb{J} = A \cup B$.

3.4 Job Clustering

Definition 23 *Let $\mathbb{J}$ be a set of jobs of a 2-PFS instance, where $t_{1,j}$ and $t_{2,j}$ represent the processing times of a job $j \in \mathbb{J}$ on the first and second machines, respectively. A subset of jobs $J \subseteq \mathbb{J}$ is said a* ***cluster*** *if for all jobs $a, b \in J$, $\max_{a \in J} t_{1,a} \leq \min_{b \in J} t_{2,b}$.*

In other words, if a set of jobs belong to the same cluster then the maximum processing time of these jobs on the first machine is not greater than the minimum processing time of the same jobs on the second machine.

Johnson's algorithm permits finding an optimal permutation schedule for the 2-PFS problem in $O(n \log n)$ time. In the next sections it will be proved that the same task can be accomplished in $O(n)$ time if the original job set is a cluster. In order to prove this result we first define a specific type of permutations, named as *trivial* permutations. In the sequence of this work, it will be demonstrated

that this family of permutations, when applied to a 2-PFS instance forming an unique cluster, lead to optimal solutions.

Definition 24 *Let $\mathbb{J}$ be a set of jobs of a 2-PFS instance. A permutation schedule $\pi : \{1, \ldots, |\mathbb{J}|\} \mapsto \mathbb{J}$ is said **trivial** if and only if $\pi(1) = \arg\min_{j \in \mathbb{J}} t_{1,j}$.*

Following the previous definition, the job located at the first position of a *trivial* permutation is that of minimum processing on machine 1. If more than one job meet this criterion, any can be chosen arbitrarily. Therefore, given a 2-PFS instance of n jobs it is possible to obtain a *trivial* permutation in $O(n)$ time. Finally, every instance of n jobs admits at least $(n-1)!$ *trivial* permutations.

The next lemma determines a closed form to the makespan resulting from a *trivial* permutation over a cluster:

Lemma 25 *Let $\mathbb{J}$ be a set of n jobs from a 2-PFS instance forming a cluster and makespan(π) be the makespan obtained by applying a trivial permutation schedule $\pi : \{1, 2, \ldots, |\mathbb{J}|\} \mapsto \mathbb{J}$ on $\mathbb{J}$. Then, makespan $(\pi) = \min_{j \in \mathbb{J}} t_{1,j} + \sum_{j \in \mathbb{J}} t_{2,j}$.*

Proof: By the recursive definition of makespan, the completion time of job $\pi(k)$ on the second machine is:

$$C(2, \pi(k)) = \max\{C(1, \pi(k)), C(2, \pi(k-1))\} + t_{2,\pi(k)} \tag{15}$$

On the first machine, in particular, the following equation holds:

$$C(1, \pi(k)) = \sum_{i=1}^{k} t_{1,\pi(i)} \tag{16}$$

Let us prove now by induction on k that $C(2, \pi(k))$ can be rewritten as:

$$C(2, \pi(k)) = \min_{i=1}^{k} t_{1,\pi(i)} + \sum_{i=1}^{k} t_{2,\pi(i)} \tag{17}$$

The basis of the induction is for $k = 1$. Taking equation 16 and remembering that π is a *trivial* permutation, i.e., $\pi(1)$ is the job with the lowest processing time on the first machine, follows that:

$$C(2, \pi(1)) = C(1, \pi(1)) + t_{2,\pi(1)} = t_{1,\pi(1)} + t_{2,\pi(1)} = \min_{i=1}^{k} t_{1,\pi(i)} + \sum_{i=1}^{k} t_{2,\pi(i)} \tag{18}$$

In order to prove the inductive step, let us consider now, by the induction hypothesis, that equation 17 is valid for all jobs belonging to positions 1 to $k-1$ in the permutation π. Applying equations 16 and 17 on 15:

$$C(2,\pi(k)) = \max\left\{\sum_{i=1}^{k} t_{1,\pi(i)}, \sum_{i=1}^{k-1} t_{2,\pi(i)} + \min_{i=1}^{k} t_{1,\pi(i)}\right\} + t_{2,\pi(k)}. \quad (19)$$

By rearranging the terms of equation 19:

$$C(2,\pi(k)) = \max\left\{\sum_{i=2}^{k} t_{1,\pi(i)} + t_{1,\pi(1)}, \sum_{i=1}^{k-1} t_{2,\pi(i)} + t_{1,\pi(1)}\right\} + t_{2,\pi(k)} \quad (20)$$

Once the set of jobs $\mathbb{J}$ is a cluster, follows that $t_{1,\pi(k)} \leq t_{2,\pi(l)}$ for all $1 \leq k, l \leq n$. Therefore, $\sum_{i=2}^{k} t_{1,\pi(i)} \leq \sum_{i=1}^{k-1} t_{2,\pi(i)}$. Hence, $C(2,\pi(k)) = t_{1,\pi(1)} + \sum_{i=1}^{k} t_{2,\pi(i)}$, what proves the inductive step. ■

Lemma 25 states that on a `2-PFS` instance represented by a job cluster, the makespan resulting from a *trivial* permutation schedule is always equal to the sum of the processing times on the second machine increased by the minimum processing time of a job on the first machine. The forthcoming proposition establishes that this value is a lower bound for the makespan of an optimal permutation schedule.

Lemma 26 *Let $\mathbb{J}$ be a set of n jobs from a `2-PFS` instance consisting of a cluster and $\pi^\star : \{1,2,\ldots,|\mathbb{J}|\} \mapsto \mathbb{J}$ be one of its optimal permutation schedules. Then, makespan $(\pi^\star) \geq \min_{j\in\mathbb{J}} t_{1,j} + \sum_{j\in\mathbb{J}} t_{2,j}$*

Proof: We shall demonstrate the following stronger statement. For all $k \in \{1,2,\ldots,n\}$:

$$C(2,\pi^\star(k)) \geq \min_{i=1}^{k} t_{1,\pi^\star(i)} + \sum_{i=1}^{k} t_{2,\pi^\star(i)} \quad (21)$$

The proof proceeds by means of induction on k. The basis of the induction is for $k = 1$. From the makespan definition follows that $C(2,\pi^\star(1)) \geq C(1,\pi^\star(1)) + t_{2,\pi^\star(1)} = t_{1,\pi^\star(1)} + t_{2,\pi^\star(1)}$. Once $t_{1,\pi^\star(1)} \geq \min_{i=1}^{k}\left\{t_{1,\pi^\star(k)}\right\}$ the basis of induction is true. In order to demonstrate the inductive step, let us consider by the inductive hypothesis that the following inequality holds: $C(2,\pi^\star(k-1)) \geq \min_{i=1}^{k-1}\left\{t_{1,\pi^\star(i)}\right\} + \sum_{i=1}^{k-1} t_{2,\pi^\star(i)}$. From the makespan recurrence, $C(2,\pi^\star(k)) = \max\{C(1,\pi^\star(k)), C(2,\pi^\star(k-1))\} + t_{2,\pi^\star(k)} \geq C(2,\pi^\star(k-1)) + t_{2,\pi^\star(k)}$. Once the induction hypothesis is valid: $C(2,\pi^\star(k)) \geq \min_{i=1}^{k-1}\left\{t_{1,\pi(i)^\star}\right\} + \sum_{i=1}^{k-1} t_{2,\pi^\star(k)} + t_{2,\pi^\star(k)}$. Therefore, the inductive step was proved and the inequality 21 holds. ■

The results obtained so far lead to the following Theorem:

Theorem 27 *Let $\mathbb{J}$ be a set of n jobs from a* `2-PFS` *instance consisting of a cluster. A trivial permutation is an optimal permutation schedule for $\mathbb{J}$ and can be obtained in $\Theta(n)$ time.*

Proof: Once by Lemmas 25 and 26 the makespan resulting from a trivial permutation π is, respectively, an upper bound and a lower bound for the optimal permutation schedule, π is optimal. From the definition of *trivial permutation* follows that π can be determined in $\Theta(n)$ steps by a simple reduction to the problem of finding the minimum from a list of n elements. ■

3.5 Cluster Decomposition

Last section introduced the concept of job cluster, proving that it is easy to find optimal permutation schedules for the `2-PFS` problem if all jobs belong to the same cluster. In this section we extend the study of optimal permutation schedules to job sets consisting of $\kappa \geq 2$ clusters.

Definition 28 *Let $\mathbb{J}$ be a* `2-PFS` *instance formed by n jobs and $\pi : \{1,\ldots,n\} \mapsto \mathbb{J}$ be a permutation schedule. The* ***turning point*** *of π is a job $\pi(k)$, $k \in \{1,\ldots,n\}$, such that $makespan(\pi) = \sum_{i=1}^{k} t_{1,\pi(i)} + \sum_{j=k}^{n} t_{2,\pi(j)}$.*

Lemma 29 *Let $\pi : \{1,\ldots,n\} \mapsto \mathbb{J}$ be a a permutation schedule. Let $\pi(k)$ be a turning point in π. If $t_{1,\pi(k)} \leq t_{1,\pi(i)}$ for every $i \in \{k, k+1,\ldots,n\}$, then π is an optimal permutation schedule for $\mathbb{J}$.*

Proof: Once $\pi(k) \leq \pi(i)$, for every $i \in \{k, k+1,\ldots,n\}$, and $\pi(k)$ is a turning point in π, $makespan(\pi)$ can be rewritten as the following summation:

$$makespan(\pi) = \sum_{i=1}^{k-1} t_{1,\pi(i)} + \min_{j=k}^{n} t_{1,\pi(j)} + \sum_{j=k}^{n} t_{2,\pi(j)} \tag{22}$$

Suppose by means of contradiction that π is not optimal. Therefore, there exists a permutation $\sigma : \{1,\ldots,n\} \mapsto \mathbb{J}$, $\sigma \neq \pi$ such that $makespan(\sigma) < makespan(\pi)$.

Let $\sigma(\gamma)$ be a turning point in σ. By the definition of turning point, $makespan(\sigma)$ can be rewritten as:

$$makespan(\sigma) = \sum_{i=1}^{\gamma} t_{1,\sigma(i)} + \sum_{j=\gamma}^{n} t_{2,\sigma(j)} \tag{23}$$

Let $\alpha \in \{1, \ldots, n\}$ be the minimum possible value such that the job represented by $\sigma(\alpha)$ is positioned on permutation π at a position less than or equal to k. In other words, $\alpha = \min_{j=1}^{n} \{\pi^{-1}(\sigma(j)) \geq k\}$. Once α is not necessarily a turning point in σ:

$$\sum_{i=1}^{\alpha} t_{1,\sigma(i)} + \sum_{j=\alpha}^{n} t_{2,\sigma(j)} \leq \sum_{i=1}^{\gamma} t_{1,\sigma(i)} + \sum_{j=\gamma}^{n} t_{2,\sigma(j)} \tag{24}$$

The expression on the left-hand of previous equation can be rearranged as:

$$\sum_{i=1}^{\alpha-1} t_{1,\sigma(i)} + t_{1,\sigma(\alpha)} + \sum_{j=\alpha}^{n} t_{2,\sigma(j)} \tag{25}$$

Once $\alpha = \min_{j=1}^{n} \{\pi^{-1}(\sigma(j)) \geq k\}$ all the jobs from the set $\{\sigma(1), \ldots, \sigma(\alpha-1)\}$ belong to the set $\{\pi(1), \ldots, \pi(k-1)\}$. By using this property and remembering that $t_{1,i} \geq t_{2,i}$ for all $i \in \mathbb{J}$, one can conclude two facts. First, $t_{1,\sigma(\alpha)} \geq \min_{j=k}^{n} t_{1,\pi(j)}$. Second, the following inequality holds:

$$\sum_{i=1}^{\alpha-1} t_{1,\sigma(i)} + \sum_{j=\alpha}^{n} t_{2,\sigma(j)} \geq \sum_{i=1}^{k-1} t_{1,\pi(i)} + \sum_{j=k}^{n} t_{2,\pi(j)} \tag{26}$$

Therefore, $makespan(\pi) \leq makespan(\sigma)$ what is a contradiction the fact that π is not optimal. ■

Definition 30 *Let $\mathbb{J}$ be a 2-PFS instance formed by n jobs. A **cluster decomposition** of $\mathbb{J}$ is a family of job clusters $\{J_1, \ldots, J_m\}$ such that $\bigcup_{t=1}^{m} J_i = \mathbb{J}$ and $J_i \cap J_j = \emptyset$, $\forall J_i, J_j \subseteq \mathbb{J}$.*

Lemma 31 *Let $\mathbb{J}$ be a 2-PFS instance formed by n jobs and $\{J_1, \ldots, J_m\}$ be a cluster decomposition of $\mathbb{J}$. Let $\sigma_1, \ldots, \sigma_m$ be trivial permutations for the job clusters $J_1, \ldots, J_m$, respectively. Let $\pi : \{1, \ldots, n\} \mapsto \mathbb{J}$ be a permutation schedule for $\mathbb{J}$ defined as $\pi = \sigma_1 \circ \ldots \circ \sigma_m$. Then, there is a turning point $\sigma_i(1)$ in one of the trivial permutations $\sigma_1, \ldots, \sigma_m$ such that $\sigma_i(1)$ is a turning point in π;*

Proof: Let $\pi(k)$ be a turning point in π. From the definition of turning point, $makespan(\pi)$ can be calculated as follows:

$$makespan(\pi) = \sum_{i=1}^{k} t_{1,\pi(i)} + \sum_{j=\pi}^{n} t_{2,\sigma(j)} \tag{27}$$

Since $J_1, \ldots, J_m$ is a cluster decomposition of $\mathbb{J}$, it is true that $\pi(k) = \sigma_p(k')$ for some $p \in \{1, \ldots, m\}$ and some $k' \in \{1, \ldots, |J_p|\}$.

Once $\pi = \sigma_1 \circ \ldots \circ \sigma_m$, equation 27 can be rewritten as:

$$makespan(\pi) = \sum_{i=1}^{k_1} t_{1,\pi(i)} + \sum_{i'=1}^{k'} t_{1,\sigma_t(i')} + \sum_{j'=k'}^{|J_p|} t_{2,\sigma_t(j')} + \sum_{j=k_2}^{n} t_{2,\pi(j)} \tag{28}$$

Since σ_p is a trivial permutation of a job cluster, $\sigma_p(1)$ is a turning point in σ_p. As consequence, $makespan(\sigma_p)$ can be determined by the expression:

$$makespan(\sigma) = t_{1,\sigma_p(i)} + \sum_{j=1}^{|J_p|} t_{2,\sigma_p(j)} \tag{29}$$

From the definition of turning point, the following inequality holds:

$$t_{1,\pi(i)} + \sum_{j=1}^{|J_p|} t_{2,\sigma_p(j)} \geq \sum_{i'=1}^{k'} t_{1,\sigma_p(i')} + \sum_{j'=k'}^{|J_p|} t_{2,\sigma_p(j')} \tag{30}$$

Hence, $makespan(\pi)$ can be rewritten as:

$$makespan(\pi) = \sum_{i=1}^{k_1} t_{1,\pi(i)} + \sum_{i'=1}^{1} t_{1,\sigma_p(i')} + \sum_{j'=1}^{|J_p|} t_{2,\sigma_p(j')} + \sum_{j=k_2}^{n} t_{2,\pi(j)} \tag{31}$$

Therefore, $\sigma_p(1)$ is a turning point in π.

■

Theorem 32 *Let $\mathbb{J}$ be a 2-PFS instance, $\{J_1, \ldots, J_m\}$ be a cluster decomposition of $\mathbb{J}$ and $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ be trivial permutations over $\{J_1, \ldots, J_m\}$, respectively. Consider now a total order $\preceq$ of Σ, defined as: $\sigma_i \preceq \sigma_j$ if and only if $t_{1,\sigma_i(1)} \leq t_{1,\sigma_j(1)}$. Let $\pi : \mathbb{J} \mapsto \mathbb{J}$ be a permutation obtained by the concatenation of the trivial permutations from Σ following the total order $\preceq$. In other words, $\pi = \sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_m$, where $\sigma_1 \preceq \sigma_2 \preceq \ldots \preceq \sigma_m$. Then π is an optimal permutation schedule for $\mathbb{J}$.*

Proof: By Lemma 31 there is a trivial permutation $\sigma_i \in \Sigma$ such that $\sigma_i(1)$ is a turning point of both π and σ_i. Let $\pi(k) = \sigma_i(1)$ be this job. Once $\sigma_1, \ldots, \sigma_m$ are trivial permutations and $\sigma_1 \preceq \sigma_2 \preceq \ldots \preceq \sigma_m$, follows that $t_{1,\sigma_i(1)} \leq t_{1,\sigma_{i'}(j)}$ for all $i \leq i' \leq m$ and $1 \leq j \leq |J_{i'}|$. Since $\pi = \sigma_1 \circ \sigma_2 \circ \ldots \circ \sigma_m$, we have that $t_{1,\pi(k)} \leq t_{1,\pi(k')}$, for all $k' \in \{k, k+1, \ldots, n\}$. Therefore, by Lemma 29, π is an optimal permutation schedule for $\mathbb{J}$. ■

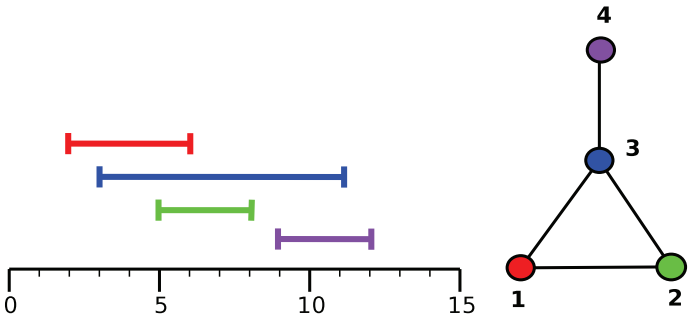
As stated by Theorem 32, it is possible to determine an optimal permutation schedule for a 2-PFS instance by a concatenation of trivial permutations, each one corresponding to a job cluster. However, these trivial permutations must be sorted on a specific way before being concatenated. In particular, the key used for this non-decreasing sorting is the processing time of the first job of each permutation on the first machine. Since, from Theorem 27, a trivial permutation of a job cluster can be determined in linear time, we proceed on the investigation of efficient algorithms to decompose a 2-PFS instance into job clusters. In the next section we explore the concept of interval graphs, one of the most fundamental graph classes, which is strongly related to the 2-PFS problem.

3.6 Interval Graphs

In this section we introduce the concept of interval graph and review some basic results concerning this graph class.

Definition 33 *Let $\mathbb{R}$ denote the set of Real numbers where a and b are two of its elements, with $a \leq b$. A subset $[a, b] = \{p \in \mathbb{R} | a \leq p \leq b\}$ of $\mathbb{R}$ is said an **interval**. Furthermore, it is said that the two intervals $[a_1, b_1]$ and $[a_2, b_2]$ **overlap** if $b_1 \geq a_2$ and $b_2 \geq a_1$.*

Definition 34 *Let $I = \{[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\}$ be a finite set of intervals. An **interval graph** $G = (V, E)$ is the undirected graph created from the set I in which there is a vertex $u \in V$ for each interval $[a_1, u_2] \in I$ and there is an edge $(u, v) \in E$ for each pair of intervals $[a_u, b_u] \in I$ and $[a_v, b_v] \in I$ that overlap. In this case, G is said the **underlying graph** of interval I.*



3.1(a): Set of intervals

3.1(b): Interval graph

Figure 3.1: Example of an interval graph

Figure 3.1 shows an example of an interval graph. This example corresponds to a set of intervals $I = \{[2,7],[3,10],[5,8],[9,12]\}$ and an underlying interval graph $G = (V, E)$, where $V = \{1,2,3,4\}$ and $E = \{(1,2),(1,3),(2,3),(3,4)\}$.

The concept of interval graph was introduced independently by Hajos and Benzer. [22]. In 1957, Hajos proposed the problem of characterizing interval graphs. In 1959, the molecular biologist Seymour Benzer asked a related question during an investigation of the topology of genes. The first characterization of interval graphs is due to Lekkerkerker and Boland [22]:

Theorem 35 *An undirected graph $G = (V, E)$ is an interval graph if and only if the following two conditions are satisfied:*
(i) G is a chordal graph, i.e., every cycle of length strictly greater than 3 possesses an edge joining two nonconsecutive vertices of the cycle.
(ii) Any three vertices of G can be ordered in such a way that every path from the first to the third vertex passes through a neighbor of the second vertex.

Interval graphs can be also characterized as those graphs whose clique matrices satisfy the consecutive 1's property for columns. A clique matrix is a matrix obtained from a graph G by its decomposition into maximal cliques, with a row for every maximal clique in G and a column for every vertex in G. A cell (i, j) in a clique matrix has an entry 1 if the maximal clique i has the vertex j and 0 otherwise. A matrix whose entries are zeros and ones is said to have the consecutive 1's property for columns if its rows can be permutated in such a way that the 1's in each column occur consecutively. This characterization is due to Fulkerson and Gross [22]. Finally, Booth and Leuker (1965) proved that interval graphs can be recognized in $O(|V| + |E|)$ time using the PQ-tree data structure, introduced by the same authors [22].

3.7 Cliques and Covers

In this section we introduce the concept of clique covering of a graph and review three algorithms for solving the minimum clique covering problem on interval graphs.

(a) Definitions

Definition 36 *Let $G = (V, E)$ be a graph. A **clique** on G is a graph $K_n = (V', E')$ such that $V' \subseteq V, E' \subseteq E$, $|V'| = n$ and for all pairs of vertices $u, v \in V'$ there is an edge $(u, v) \in E'$. In this case, K_n is said a clique of size n.*

From the definition of clique follows that, an interval graph corresponding to a set of intervals I on the real line is a clique if and only if all of its pairs of vertices overlap, i.e., $\max_{[a_i,b_i]\in I}\{a_i\} \leq \min_{[a_i,b_i]\in I}\{b_i\}$. Due to this fact, recognizing if a graph is a clique is computationally little time consuming for interval graphs in comparison to ordinary graphs. If the edges of the interval graph are implicitly defined by a set of n intervals, it is possible to determine if this graph is a clique in $\Theta(|V|)$ time. However, determining if an ordinary graph is a clique requires at least $\Omega(|V| + |E|) = \Omega(|V|^2)$ steps in the worst case.

The decision problem of, given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, determining if G has a clique of size at least k is *strongly NP-hard* [19]. However, the same question can be solved in polynomial time if G is an interval graph [22].

Definition 37 *Let $G = (V, E)$ be a graph. A **clique covering** of G is a set of graphs $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots, G_p = (V_p, E_p)$, such that $\bigcup_{i=1}^{p} V_i = V$ and, for all $1 \leq i \leq p$, the graph G_i is a clique.*

Figure 3.2 shows an example of a clique covering of a graph.

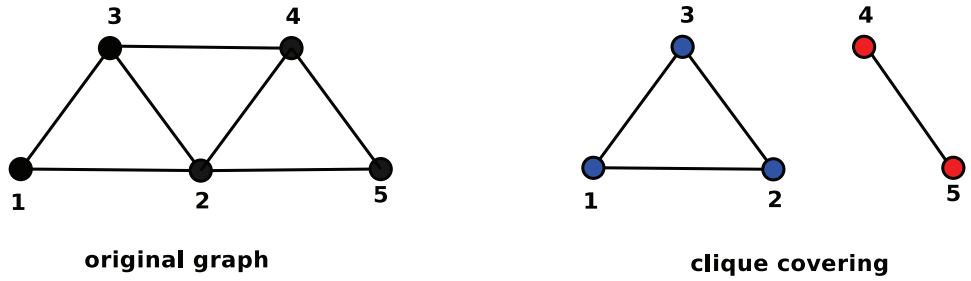


Figure 3.2: A clique covering of a graph

The `Clique Covering Problem` is the decision problem of, given a graph G and a positive integer $k \leq |V|$, determining if there is a set of k cliques or less whose union is the entire vertex set of G. If $k \leq 2$ the `Clique Covering Problem` can be solved in polynomial time by a reduction to the maximum cardinality matching problem. However, for all fixed $k \geq 3$ this problem becomes NP-complete [19]. The equivalent optimization problem, named `Minimum Clique Covering Problem`, asks for a clique covering that minimizes k. In graph theory, the minimum integer k for which there exists a clique covering of a graph $G = (V, E)$ is named the *clique covering number* of G, being denoted by $\kappa(G)$ [4].

(b) Algorithms

Three algorithms, of distinct time complexities, are known for solving the minimum clique covering problem on interval graphs. The first one is due to Gavril [21], who provided a $\Theta(|V| + |E|)$ algorithm to solve this problem for the class of chordal graphs. Once that by Theorem 35 every interval graph is a chordal, the `Clique Covering Problem` can be solved in linear time if G is an interval graph. This algorithm proceeds by listing all maximal cliques of the graph (cliques not contained in other cliques) using a *perfect elimination scheme*, which can be implemented by a breadth-first search [22].

The second algorithm is due to Gupta, Lee and Leung [29] and works with an interval graph given in the form of a family of intervals. They show that a maximum independent set, a maximum clique and a minimum clique covering can be found in $\Theta(|V| \log |V|)$. The algorithm works in two stages. In the first stage, the intervals are sorted by it right endpoints in $\Theta(|V| \log |V|)$ time. In the second stage, a minimum clique covering is constructed in $\Theta(|V|)$ time by scanning the intervals following such ordering, always trying to insert the current interval on the current clique, in order to increase the its size. If this is not possible, a new clique is created containing the current interval, becoming the current clique.

Before presenting the third algorithm, some additional definitions become necessary. An *independent set* of a graph $G = (V, E)$ is a subset of vertices $I \subset V$ such that for all pairs of vertices $u, v \in I$ we have $(u, v) \notin E$. An independent set I_1 is said *maximum* if there is no independent set $I_2 \in V$ such that $|I_2| \geq |I_1|$. In this case, $|I_1|$ is said the *independence number* of G, being denoted by $\alpha(G)$. A classical result from Gallai [57] states that, if G is an interval graph then its independence and clique covering numbers are equal, i.e, $\alpha(G) = \kappa(G)$.

The third algorithm is based on the work of Snoeyink [59] and also works with an interval graph given in the form of a family of intervals. In this work, it was demonstrated an $O(|V| \log \alpha(G))$ divide-and-conquer algorithm that produces a maximum independent set of an interval graph with independence number equal to $\alpha(G)$. However, it is not hard to extend it to obtain a minimum clique covering for the same graph in $O(|V| \log \kappa(G))$ by using the theorem from Gallai [57]. This method can be implemented in $O(|V| \log \kappa(G))$ by using a binary search tree to store the intervals of the independent set. In this case, the query used for inserting a vertex on its corresponding clique or creating a

new clique can be executed in $O(\log \kappa(G))$ time. The algorithm from Snoeyink is more efficient than the algorithm from Gupta et al., since, by the definition of clique covering, $\kappa(G) \leq |V|$. Furthermore, it is an optimal algorithm due to a well-known lower bound of $\Omega(n \log k)$ comparisons for sorting a list of n elements from which only k are distinct[38]. Table 3.1 summarizes the three algorithms introduced in this section for determining minimum clique coverings on interval graphs.

Author	Year	Technique	Time Complexity
Gavril	1972	Breadth-First Search	$\Theta(\|V\| + \|E\|)$
Gupta et al.	1982	Sorting	$\Theta(\|V\| \log \|V\|)$
Snoeyink	2007	Divide-and-Conquer	$\Theta(\|V\| \log \kappa(G))$

Table 3.1: Clique covering algorithms for interval graphs.

3.8 2-PFS and Interval Graphs

In this section we demonstrate how to reduce the problem of finding a cluster decomposition of a set of jobs to the problem of determining a clique covering for an equivalent interval graph.

Let $\mathbb{J}$ be a set of jobs representing a `2-PFS` instance. Let us define an *equivalent interval graph* G given in the form of a family of intervals. More precisely, for every job $j \in \mathbb{J}$ with processing times $t_{1,j}$ and $t_{2,j}$ on first and second machines, we associate an interval $[a, b] \in G$ such that $a = t_{1,j}$ and $b = t_{2,j}$. Once that, as proved at section 3.3, all jobs of a `2-PFS` instance can be assumed as having its processing times on the first machine less then or equal to its processing times on the second machine, this interval graph can always be constructed. Next lemma establishes that finding a job decomposition of a `2-PFS` instance is equivalent to determine a clique covering of its equivalent interval graph.

Lemma 38 *Let $\mathbb{J}$ be a `2-PFS` instance and $G = (V, E)$ be its equivalent interval graph. Then, $\mathbb{J}$ admits a decomposition into k clusters if and only if G has clique covering of size k.*

Proof: The proof of necessity is constructive. Suppose that $\mathbb{J}$ admits a decomposition into k clusters. Let $J = \{j_1, \ldots, j_t\}$ be a subset of jobs in $\mathbb{J}$ forming one of these clusters. Let $H = \{[a_1, b_1], \ldots, [a_t, b_t]\}$ be a subset of intervals from G such that $[a_i, b_i] \in H$ is equivalent to $j_i \in J$, for all $1 \leq i \leq t$.

From the definition of job cluster follows that $\max_{1\leq i\leq t} t_{1,j_i} \leq \min_{1\leq i\leq t} t_{2,j_i}$. Therefore, for all vertices $[a_i, b_i] \in H$ it is true that $\max_{1\leq i\leq t} a_i \leq \min_{1\leq i\leq t} b_i$. Hence, H is a clique. The proof of sufficiency follows similar arguments. ■

Let us proceed now by reviewing some important results from last sections. At the end of the section 3.3 it was proved that its is possible to determine an optimal permutation schedule for a `2-PFS` instance by a concatenation of trivial permutations, each one corresponding to a job cluster. However, as stated by Theorem 32, such trivial permutations must be sorted in non-decreasing way before being concatenated. The key used for this non-decreasing sorting is the processing time of the first job of each permutation on the first machine. Therefore, any efficient comparison based algorithm can be used for sorting such trivial permutations. Furthermore, from theorem 27, a trivial permutation can be easily obtained in linear time.

The question that kept unanswered until the beginning of this section was how to decompose the original set of jobs into job clusters. In order to provide an answer to this question, the concept of interval graph was introduced at section 3.7. The minimum clique covering problem was also defined at section 3.7, and three polynomial time algorithms with distinct time complexities were reviewed for this problem. Finally, lemma 38 demonstrated that the cluster decomposition problem of a `2-PFS` instance admits a linear time reduction to the clique covering problem of an equivalent interval graph.

As consequence of the previous results obtained, we present now an algorithm for finding an optimal permutation schedule of a `2-PFS` instance in $\Theta(n \log \kappa)$, where κ is the minimum number of job clusters necessary to decompose the original `2-PFS` instance.

Algorithm 3: Fast `2-PFS` algorithm

Input : Set $\mathbb{J}$ of n jobs,
Processing times matrix $P \in \Re^{+}_{2\times\mathbb{J}}$.
Output: permutation schedule $\pi : \{1, \ldots, n\} \mapsto \mathbb{J}$.
begin
Let A be the subset of jobs $j \in \mathbb{J}$ such that $t_{1,j} \leq t_{2,j}$;
Let $B \longleftarrow \mathbb{J} \backslash A$;

Create an interval graph G_A equivalent to the job set A ;
Create an interval graph G_B equivalent to the job set B ;

Find a minimum clique covering $C = \{c_1, \ldots, c_t\}$ of G_A ;
Find a minimum clique covering $D = \{d_1, \ldots, d_s\}$ of G_B ;

Sort C in non-decreasing order using the minimum left endpoint of each clique c_i as the key ;

Sort D in non-decreasing order using the minimum left endpoint of each clique d_i as the key ;

$\pi_A \longleftarrow \emptyset$;
for *every clique* c_i *in the sorted set* C **do**
  $\varphi_i \longleftarrow$ trivial permutation of jobs equivalent to c_i ;
  $\pi_A \longleftarrow \pi_A \circ \varphi_i$;

$\pi_B \longleftarrow \emptyset$;
for *every clique* d_i *in the sorted set* D **do**
  $\sigma_i \longleftarrow$ trivial permutation of jobs equivalent to d_i ;
  $\pi_B \longleftarrow \pi_B \circ \sigma_i$;

$\pi \longleftarrow \pi_A \circ \pi_B$;
return π ;
end

Theorem 39 *Fast 2-PFS algorithm generates an optimal permutation schedule for the 2-PFS problem and can be implemented in* $\Theta(n \log \kappa)$ *time, where* κ *is the minimum number of job clusters necessary to decompose* $\mathbb{J}$.

Proof: The correctness proof of `Fast 2-PFS algorithm` comes directly from theorem 32 and lemma 38. Let us analyze now the time complexity of the algorithm proposed. The algorithm starts by splitting $\mathbb{J}$ into sets A and B, what can done in $\Theta(n)$ time. The creation of the interval graph G_A can be done by associating an interval to each job of A, as demonstrated at the beginning of this section. The creation of G_B from set B is similar. For every job $j \in \mathbb{J}$ we define an interval $[u, v] \in G_B$ with $u = t_{2,j}$ and $v = t_{1,j}$. Since G_A and G_B are represented by a family of intervals, their edge sets will be implicitly defined. Hence, G_A and G_B can be determined in $\Theta(n)$ time.

Minimum clique coverings for G_A and G_A can be obtained in $\Theta(n \log \kappa)$ by applying the algorithm of Snoeyink, introduced at section 3.7. The choice of this algorithm is related to its lower time complexity in comparison to the algorithms of Gavril and Gupta et al. Applying the algorithm of Gavril, in particular, would result on a time complexity of $\Theta(n^2)$ since its time complexity considers the number of edges in G, which can be up to n^2. The use of the algorithm of Gupta et al implies on a time complexity of $\Theta(n \log n)$, which does not result on an improvement over the classical `2-PFS` algorithm of Johnson.

The minimum clique coverings C and D can be sorted $\Theta(\kappa \log \kappa)$ using any optimal comparison based sorting algorithm [38]. The keys used for sorting these sets of cliques are the minimum left endpoints of each clique. All trivial permutations φ_i and σ_i can be determined in $\Theta(n)$ time, as proved in section 3.4. Permutations π_A and π_B can also be determined in $\Theta(n)$ time, as well as the final permutation schedule π. Therefore the `Fast 2-PFS algorithm` can be executed in $\Theta(n \log \kappa)$ time. ■

3.9 Conclusion

This chapter introduces a new and faster algorithm for solving the `2-PFS` problem. This algorithm runs in $\Theta(n \log \kappa)$ time complexity, where κ is the clique covering number of an interval graph equivalent to the `2-PFS` instance. To the best of our knowledge this is the faster algorithm already obtained for the `2-PFS` problem.

The `Fast 2-PFS algorithm` is composed by five stages. Stage I split the set of jobs $\mathbb{J}$ into two disjoint sets, A and B. Stage II transforms A and B to equivalent interval graphs, denoted by G_A and G_B, respectively. This reduction runs in linear time. Stage III uses an $\Theta(n \log \kappa)$ algorithm to find minimum clique coverings C_A and C_B in the interval graphs G_A and G_B, respectively. The parameter κ denotes the minimum number of *job clusters* necessary to decompose the original job set, which is equal to the sum of the clique covering numbers of the interval graphs G_A and G_B. Stage IV order the κ cliques in C_A and C_B in $\Theta(\kappa \log \kappa)$ time using any efficient comparison based sorting algorithm. Stage V determines the final permutation schedule in $\Theta(n)$ time using the sorted clique covers. Therefore, the `Fast 2-PFS algorithm` can be executed in $\Theta(n \log \kappa)$.

It is important to stress that this new algorithm is not based on faster *sorting* methods. Our algorithm uses a classical $\Theta(n \log n)$ comparison based sorting

method to achieve the claimed $\Theta(n \log \kappa)$ time complexity. In particular our results are based on the fact that is possible to decompose the jobs into κ disjoint subsets and order κ elements to achieve an optimal solution.

Furthermore, we prove that the time complexity of algorithm `Fast 2-PFS` problem is dominated by the time complexity of finding an equivalent minimum clique covering problem in interval graphs. This connection between `2-PFS` and minimum clique covering on interval graphs is, from the best of our knowledge, completely new.

Finally, the theoretical results supporting this new algorithm provide a more precise characterization of optimal permutation schedules for `2-PFS` instances. As Pinedo [49] said "*Johnson's schedules are by no means the only schedules that are optimal for 2-PFS. The class of optimal schedules appears to be hard to characterize and data dependent* ". One of the contributions of this work, in particular, is to provide a better understanding about the class of optimal permutation schedules for the `2-PFS` problem. In particular, it was proved that by ordering any cluster decomposition of a `2-PFS` instance we create an optimal solution. This result is a generalization of the results obtained by Johnson, in which there were always n disjoint clusters consisting of a single job.