NRC Publications Archive Archives des publications du CNRC

A divide and conquer-based greedy search for two-machine no-wait job shop problems with makespan minimisation

Zhu, Jie; Li, Xiaoping; Shen, Weiming

This publication could be one of several versions: author's original, accepted manuscript or the publisher's version. / La version de cette publication peut être l'une des suivantes : la version prépublication de l'auteur, la version acceptée du manuscrit ou la version de l'éditeur.

For the publisher's version, please access the DOI link below. / Pour consulter la version de l'éditeur, utilisez le lien DOI ci-dessous.

Publisher's version / Version de l'éditeur:

<https://doi.org/10.1080/00207543.2011.582185>

International Journal of Production Research, 50, 10, pp. 2692-2704, 2011-07-12

NRC Publications Record / Notice d'Archives des publications de CNRC:

<https://nrc-publications.canada.ca/eng/view/object/?id=b9e2422a-775a-48a0-bae9-77301e6e1d6f>

<https://publications-cnrc.canada.ca/fra/voir/objet/?id=b9e2422a-775a-48a0-bae9-77301e6e1d69>

Access and use of this website and the material on it are subject to the Terms and Conditions set forth at

<https://nrc-publications.canada.ca/eng/copyright>

READ THESE TERMS AND CONDITIONS CAREFULLY BEFORE USING THIS WEBSITE.

L'accès à ce site Web et l'utilisation de son contenu sont assujettis aux conditions présentées dans le site

<https://publications-cnrc.canada.ca/fra/droits>

LISEZ CES CONDITIONS ATTENTIVEMENT AVANT D'UTILISER CE SITE WEB.

Questions? Contact the NRC Publications Archive team at

PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca. If you wish to email the authors directly, please see the first page of the publication for their contact information.

Vous avez des questions? Nous pouvons vous aider. Pour communiquer directement avec un auteur, consultez la première page de la revue dans laquelle son article a été publié afin de trouver ses coordonnées. Si vous n'arrivez pas à les repérer, communiquez avec nous à PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca.

A Divide and Conquer-based Greedy Search for Two-machine No-wait Job Shop Problems with Makespan Minimization

Jie Zhu^{1,2*}, Xiaoping Li^{1,2,3*}, Weiming Shen³

1 School of Computer Science and Engineering, Southeast University, Nanjing, China

2 Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education

3 Centre for Computer-assisted Construction Technologies National Research Council, London, Ontario, Canada

This paper addresses a two-machine no-wait job shop problem with makespan minimization. It is well known that this problem is strongly NP-hard. A divide-and-conquer approach (*DC* for short) is adopted to calculate the optimal timetable of a given sequence. It decomposes the given sequences into several independent parts and conquers them separately. A timetable enhancing method is introduced to further improve the timetable obtained by *DC*. It constructs a set of flow shop type jobs based on the result from *DC* and calculates the best timetable for these newly constructed jobs by the well-known Gilmore and Gomory method (*GG* for short). An efficient greedy search is proposed by integrating *DC* with *GG* to search for the best sequence. Experimental results show that the proposed algorithm can find the optimal solutions for 96% of the randomly generated test instances on average.

Keywords: scheduling, no-wait, job shop, two-machine

1. Introduction

In this paper, the no-wait job shop problem with makespan minimization is addressed in a two-machine environment. The problem can be denoted as $J_2 | nwt | C_{\max}$ according to the notation by Graham et al. (1979). It is known that the two-machine job shop problem $J_2 // C_{\max}$ has been optimally solved by Jackson's well-known $O(n \log n)$ algorithm in Jackson (1956). The considered problem is different from the classical job shop problem because of an additional no-wait constraint, i.e., consecutive operations of each job must be performed continuously without any interruption.

The traditional job shop problem is a theoretical concept rooted in many real life problems. It is well known to be NP-hard in the strong sense (Lenstra and

Rinnooy, 1979) and has been intensely investigated during past decades (Jain and Meeran, 1990, Blazewicz et al., 1996). Based on these investigations, the research has been extended to focus on more concrete cases raised from industrial applications, such as dynamical job shop problems which consider random job arrivals and machine breakdowns (Zandieh and Adibi, 2010) and flexible job shop problems which allows an operation of each job to be executed by any machine out of a set of available machines (Amiri et al., 2010, Rajkumar et al., 2010).

Such no-wait constraints usually arise spontaneously from processing environments and/or characteristics of the jobs. Typical examples are metallurgical processes where the iron has to be stricken literally while hot, chemical processes with unstable intermediate products, or simply the absence of intermediate storage capacity. Such job shop problems widely exist in the real world, such as in the chemical and pharmaceutical industries (Rajendran, 1994, Raaymakers and Hoogeveen, 2000), steel production (Wisner, 1972), computer systems (Reddi and Ramamoorthy, 1973), semiconductor testing facilities (Ovacik and Uzsoy, 1997), surgical processes (Dinh-Nguyen and Andreas, 2008), and production of concrete wares (Grabowski and Pempera, 2000).

Despite its relevancy and importance, little attention has been paid to the problem with a no-wait constraint from either a theoretical or computational perspective. Hall and Sriskandarajah (1996) reviewed the computational complexity of a wide variety of no-wait scheduling problems and showed that the considered problem is difficult, especially for large-size instances. It was proven to be NP-Hard in the strong sense even for two-machine cases (Sahni and Cho, 1979). Recently, even stronger statements regarding its complexity were proven in Woeginger (2004), in which the approximability of the no-wait job shop scheduling problem is investigated

and two cases were proven to be APX-Hard: (1) the case of two machines with, at most, five operations per job, and (2) the case of three machines with, at most, three operations per job. Bansal et al. (2005) showed that the problem admits a polynomial time approximation scheme if each job has, at most, two operations and the number of machines is a constant.

Some approximate algorithms have been developed for specific production systems. A decomposition approach making use of fuzzy logic and an algorithm based on a Tabu search were developed for the scheduling problem of no-wait manufacturing processes in Macchiaroli and Mole (1999). Raaymakers and Hoogeveen (2000) characterized the scheduling problems in multipurpose batch process industries as multiprocessor no-wait job shop problems with overlapping operations and proposed a simulated annealing algorithm. Ovacik and Uzsoy (1997) focused on the scheduling problems in semiconductor manufacturing and formulated the factory scheduling problem as job shop problems with a no-wait feature. They addressed time-based decomposition (rolling horizon) procedures combining a degree of global information with optimization to make dispatching decisions. All of these approximation procedures are focused on specific production environments; therefore, no results on known benchmark instances are given in these papers. Meloni et al. (2004) investigated the job shops with blocking or no-wait constraints and reported on an extensive study on the applicability of a meta-heuristic approach called rollout or pilot method. Mascis and Pacciarelli (2002) formulated the problem by means of an alternative graph and developed branch and bound algorithms. Two local search algorithms VNS (variable neighborhood search) and GASA (hybrid simulated annealing/genetic algorithm) were proposed for the considered problem by Schuster and Framinan (2003). Then they implemented a complete local search with memory

(CLM) to further improve the computational results on the benchmark instances (Framinan and Schuster, 2006). Schuster (2006) also developed a fast Tabu search for the problem. Based on his literature, we previously proposed a complete local search with limited memory (CLLM) (Zhu et al., 2009).

In this paper, a greedy search is presented for the problem $J_2 | nwt | C_{\max}$. It is composed of two components, respectively, for the timetabling problem and the sequencing problem. The timetabling problem is to find an optimal timetable for a given sequence and calculate its makespan. The sequencing problem is to search for the best sequence which has the minimal makespan. The timetabling problem can be solved by a divide and conquer approach in polynomial time complexity. By transforming the considered problem into a no-wait flow shop problem as in Liaw (2008), the obtained timetable can be enhanced by the well-known Gilmore and Gomory algorithm (Gilmore and Gomory, 1964). The greedy search is proposed for the sequencing problem.

The rest of the paper is organized as follows. In Section 2, a problem description and notations used in the paper are presented. A divide-and-conquer approach is introduced to optimally solve the timetabling problem in Section 3. In Section 4, a timetable enhanced method is proposed based on the well-known Gilmore and Gomory algorithm. The greedy search algorithm is proposed in Section 5. Finally, the experimental results are shown in Section 6, followed by conclusions in Section 7.

2. Problem Statement

This paper addresses a scheduling problem in which n jobs are processed on two machines M_1 and M_2 . Each job has to be processed on each machine exactly once by its specific processing route and any waiting time between two consecutive

operations is not allowed. A processing sequence is a permutation π of the set of jobs and denoted as vector $(\pi_{[1]}, \dots, \pi_{[n]})$. For the sake of convenience, the jobs with the first operation processed on M_1 are called the positive jobs and the jobs with a reversed processing route are called the negative jobs, as shown in Fig. 1.

A description matrix $\delta(\pi_{[i]}) = \begin{pmatrix} p_{[i]}^1 & 0 \\ 0 & p_{[i]}^2 \end{pmatrix}$ is used to describe a positive job and $\delta(\pi_{[i]}) = \begin{pmatrix} 0 & p_{[i]}^2 \\ p_{[i]}^1 & 0 \end{pmatrix}$ is used to describe a negative job. $\delta(\pi_{[i]})_{pq}$ indicates the element at the p th row and q th column of $\delta(\pi_{[i]})$.

Boolean variable $s(\pi_{[i]})$ indicates whether $\pi_{[i]}$ is positive or negative with a value *True* and *False*, respectively. Besides, some notations to be used in this paper are given in Table 1.

The considered problem is usually decomposed into two sub-problems: (1) the sequencing problem, in which a processing sequence of an optimal schedule is found for a given no-wait job shop problem, and (2) the timetabling problem, in which the optimal schedule which has the minimal makespan is found for a given processing sequence. In the paper, a polynomial algorithm based on the divide-and-conquer approach is proposed to solve the timetabling problem of a two-machine no-wait job shop problem.

3. Proposed Divide-and-conquer Method

If all jobs in a given processing sequence $\pi_{[1]}, \pi_{[2]}, \dots, \pi_{[n]}$ are negative/positive, the corresponding optimal timetable can be figured out in polynomial time using the Non-delay Timetabling method (Framinan and Schuster, 2006). However, this method cannot guarantee to get the optimal timetable for a given sequence which contains

both positive and negative jobs. Since there are only two kinds of jobs (positive and negative) in the considered problem, the timetabling problem is not so hard to solve after thoroughly investigating the job characteristics and the no-wait constraint. A divide-and-conquer method is developed to solve the timetabling problem in polynomial time for two-machine cases.

3.1 Two kinds of Super Job

Given a processing sequence $\pi_{[1]}, \pi_{[2]}, \dots, \pi_{[n]}$, the starting time of each job is suggested to follow a time order. In this paper, the time order of the second operations ($t_{\pi[1]}^2 \leq t_{\pi[2]}^2 \leq \dots \leq t_{\pi[n]}^2$) is considered and the order is called the secondary time order. The purpose of constructing an optimal timetable is to find a feasible set of starting times following the time order with the minimal makespan.

For any two consecutive jobs $\pi_{[i]}$ and $\pi_{[i+1]}$ following the secondary time order, if $s(\pi_{[i]}) = s(\pi_{[i+1]})$, $t_{\pi[i+1]} - t_{\pi[i]} \geq p_{\pi[i]}^1 + \max\{p_{\pi[i]}^2 - p_{\pi[i+1]}^1, 0\}$ (Fig. 2(a) and Fig. 2(b)) must be satisfied. When $s(\pi_{[i]}) \neq s(\pi_{[i+1]})$, $t_{\pi[i+1]} - t_{\pi[i]} = p_{\pi[i]}^1 - p_{\pi[i+1]}^1$ (Fig. 2(c) and Fig. 2(d)) or $t_{\pi[i+1]} - t_{\pi[i]} \geq p_{\pi[i]}^1 + p_{\pi[i]}^2$ (Fig. 2(e) and Fig. 2(f)) can make sure of the feasibility. The cases in Fig. 2(c) and 2(d) are called an intertwined relation and the others are called a separated relation.

The intertwined jobs $\pi_{[i]}$ and $\pi_{[i+1]}$ ($1 \leq i \leq n-1$) are taken as a super job denoted as $\langle \pi_{[i]}, \pi_{[i+1]} \rangle$. A job $\pi_{[i]}$ can only intertwine with one of its adjacencies ($\pi_{[i-1]}$ or $\pi_{[i+1]}$) in a given sequence. If a job $\pi_{[i]}$ intertwines with neither $\pi_{[i-1]}$ nor $\pi_{[i+1]}$, it is called a separated job and denoted as $\langle \pi_{[i]}, \pi_{[i]} \rangle$.

3.2 Partner sequence

In a feasible solution, a job is either a separated super job or intertwined with

one of its adjacent jobs. For two intertwined jobs which form an intertwined super job, one of them can be taken as the other one's partner. For a job forming a separated super job, its partner is itself.

Solutions of the considered problem can be codified by two vectors: a processing sequence π and its corresponding partner sequence ϖ where $\varpi_{[i]}$ is the partner of $\pi_{[i]}$ ($1 \leq i \leq n$). Initially, all items in ϖ are set to *NULL*.

Once all jobs' partners are fixed, i.e. $\forall \varpi_{[i]} \neq \text{NULL}$ ($1 \leq i \leq n$), the starting time of each job can be determined. However, a sequence can be mapped to a lot of partner sequences, while only a few of them can lead to the optimal timetable. Finding the optimal timetable for a given sequence can be equivalent to constructing the best partner sequence.

3.3 Divide and Conquer

A recursive method is introduced to divide the timetabling problem into several independent parts and solves them recursively by calling itself. The recursive scheme is the well-known divide-and-conquer approach (*DC*).

Simply dividing a sequence into several parts and conquering each part individually are not suitable for the problem in this paper, since these parts are not independent to each other. However, based on the characteristics of super jobs, the *DC* scheme is still available for the considered problem. Suppose $OPT(\pi, \varpi)$ denotes the function which outputs the optimal makespan with two parameters: a processing sequence π and its corresponding partner sequence ϖ . $\pi(p, q)$ is a subsequence containing jobs of π between two specified indices p and q , i.e. $\pi(p, q) = \pi_{[p]}, \dots, \pi_{[q]}$ ($p \leq q$).

Theorem 1. Given a sequence π and its partner sequence ϖ . If $\exists \varpi_{[k]} = \pi_{[k]} \parallel \varpi_{[k]} = \pi_{[k-1]} \quad (1 < k < n)$, $OPT(\pi, \varpi)$ can be divided into two subproblems as follows:

$$OPT(\pi, \pi^p) = OPT(\pi(1, k), \varpi(1, k)) + OPT(\pi(k, n), (\varpi(k, n))) \\ - (\max\{p_{\pi[k]}^1, p_{\varpi[k]}^1\} + \max\{p_{\pi[k]}^2, p_{\varpi[k]}^2\})$$

Proof $\varpi_{[k]} = \pi_{[k]}$ indicates $\pi_{[k]}$ is a separated super job. The Gantt chart of the optimal schedule with $\varpi_{[k]} = \pi_{[k]}$ can be split into two parts as depicted in Fig. 3, where one part is ended with $\pi_{[k]}$ and the other part is started with $\pi_{[k]}$. Apparently, these two parts are independent to each other. $\varpi_{[k]} = \pi_{[k-1]}$ indicates $\pi_{[k]}$ intertwined with $\pi_{[k-1]}$. The Gantt chart of the optimal schedule with $\varpi_{[k]} = \pi_{[k-1]}$ can be split into two parts as depicted in Fig. 4, where one part is ended with the super job $\langle \pi_{[k-1]}, \pi_{[k]} \rangle$ and the other part is started with $\langle \pi_{[k-1]}, \pi_{[k]} \rangle$. Both parts are independent to each other. Therefore, $OPT(\pi, \varpi)$ is equivalent to the sum of the optimal makespan of two parts minus their overlap. The overlap is the processing time of the super job $\langle \pi_{[k]}, \varpi_{[k]} \rangle$, i.e. $\max\{p_{\pi[k]}^1, p_{\varpi[k]}^1\} + \max\{p_{\pi[k]}^2, p_{\varpi[k]}^2\}$. $\square$

In Theorem 1, a sequence is divided into two subsequences with an assumption that one job's partner is given. However, at the very beginning, no job has its partner fixed. In order to apply Theorem 1, one job's adjacencies should be investigated. A job $\pi_{[i]}$ ($1 < i < n$) has two adjacencies: $\pi_{[i-1]}$ and $\pi_{[i+1]}$. Assume $\varpi_{[i]} = NULL$, $\varpi_{[i-1]} = NULL$ and $\varpi_{[i+1]} = NULL$. The *DC* cases are described as follows.

DC_Case (1) When $s(\pi_{[i-1]}) = s(\pi_{[i]}) = s(\pi_{[i+1]})$:

$\pi_{[i]}$ can neither intertwine $\pi_{[i-1]}$ nor $\pi_{[i+1]}$, i.e. $\pi_{[i]}$ can only be a separated super job. Therefore, $\varpi_{[i]} = \pi_{[i]}$. Theorem 1 can be applied to this case directly.

DC_Case (2) When $s(\pi_{[i-1]}) = s(\pi_{[i]})$ and $s(\pi_{[i]}) \neq s(\pi_{[i+1]})$:

For this case, $\pi_{[i]}$ can either be a separated super job or intertwine $\pi_{[i+1]}$, i.e. $\varpi_{[i]} = \pi_{[i]}$ or $\varpi_{[i]} = \pi_{[i+1]}$, where $\varpi_{[i]} = \pi_{[i+1]}$ must have $\varpi_{[i+1]} = \pi_{[i]}$ (since $\pi_{[i]}$ and $\pi_{[i+1]}$ are intertwined, $\pi_{[i+1]}$'s partner is $\pi_{[i]}$). Therefore, $OPT(\pi, \varpi) = \min\{OPT(\pi, \varpi \mid \varpi_{[i]} = \pi_{[i]}), OPT(\pi, \varpi \mid \varpi_{[i+1]} = \pi_{[i]})\}$. Theorem 1 can be applied to this case by dividing the problem into four independent subproblems.

DC_Case (3) When $s(\pi_{[i-1]}) \neq s(\pi_{[i]})$ and $s(\pi_{[i]}) = s(\pi_{[i+1]})$:

Similar to the case (2), $\varpi_{[i]} = \pi_{[i]}$ or $\varpi_{[i]} = \pi_{[i-1]}$. The problem can be divided into four independent subproblems.

DC_Case (4) When $s(\pi_{[i-1]}) = s(\pi_{[i+1]})$ and $s(\pi_{[i]}) \neq s(\pi_{[i+1]})$

$\pi_{[i]}$ can be a separated super job or intertwine with $\pi_{[i-1]}$ or $\pi_{[i+1]}$, i.e. $\varpi_{[i]} \in \{\pi_{[i-1]}, \pi_{[i]}, \pi_{[i+1]}\}$. Therefore, $OPT(\pi, \varpi) = \min\{OPT(\pi, \varpi \mid \varpi_{[i]} = \pi_{[i]}), OPT(\pi, \varpi \mid \varpi_{[i+1]} = \pi_{[i]}), OPT(\pi, \varpi \mid \varpi_{[i]} = \pi_{[i-1]})\}$. Theorem 1 can be applied to this case by dividing the problem into six independent subproblems.

The indexes of the first *NULL* item and the last *NULL* item in ϖ are denoted as f and l , respectively. For each division, job indexed by $\lfloor \frac{f+l}{2} \rfloor$ is selected to be investigated with its adjacencies according to the above four cases and this job is called the key job. At the very beginning when all jobs' partners are *NULL*, $\pi_{\lfloor \frac{1+n}{2} \rfloor}$ is the key job.

3.4 The base case of the recursion

In each division, the problem is divided into the left parts with the last one or two jobs' partner fixed and the right parts with the first one or two jobs' partner fixed. Hence, for a subsequence π , $f = 2$ when the first job is a separated super job or $f = 3$ when the first two jobs are intertwined; $l = |\pi| - 1$ when the last job is a separated super job or $l = |\pi| - 2$ when the last two jobs are intertwined. According to DC cases, $\varpi_{[f]}, \dots, \varpi_{[l]}$ are *NULL*.

If all jobs in the subsequence have their partners fixed, $f \leftarrow -1$ and $l \leftarrow -1$. The base case for the recursion is when $l - f \leq 1$ or $l = f = -1$, i.e. at most two items in ϖ are *NULL*. Then the recursion stops and the best makespan of the subsequence is returned. The base cases are described as follows.

Base case (1) $l = f = -1$

Under this case, the current subsequence is composed of super jobs. The best makespan is equal to the sum of the durations of all super jobs minus the maximal overlaps between any two consecutive super jobs. Fig. 5 shows the pseudo-code for this case.

The function $Overlap(<\pi_{[i]}, \varpi_{[i]}>, <\pi_{[j]}, \varpi_{[j]}>)$ returns the maximal overlap between super jobs $<\pi_{[i]}, \varpi_{[i]}>$ and $<\pi_{[j]}, \varpi_{[j]}>$. The maximal overlap equals the completion time of $<\pi_{[i]}, \varpi_{[i]}>$ minus the minimal starting time of $<\pi_{[j]}, \varpi_{[j]}>$. Assume $<\pi_{[i]}, \varpi_{[i]}>$ starts at $t_{[i]}$ and $<\pi_{[j]}, \varpi_{[j]}>$ starts at $t_{[i+1]}$. $<\pi_{[j]}, \varpi_{[j]}>$ is supposed to start as early as possible after $<\pi_{[i]}, \varpi_{[i]}>$, i.e., there must be no waiting time on at least one machine. Therefore, according to Fig. 6, the maximal overlap is $\max\{p_{\pi_{[i]}}^2, p_{\varpi_{[i]}}^2\} + \max\{p_{\pi_{[j]}}^1, p_{\varpi_{[j]}}^1\} - \max\{\rho_1, \rho_2\}$, where

$$\rho_1 = \max \{ \delta(\pi_{[i]})_{12}, \delta(\varpi_{[i]})_{12} \} + \max \{ \delta(\pi_{[j]})_{11}, \delta(\varpi_{[j]})_{11} \} ,$$

$$\rho_2 = \max \{ \delta(\pi_{[i]})_{22}, \delta(\varpi_{[i]})_{22} \} + \max \{ \delta(\pi_{[j]})_{21}, \delta(\varpi_{[j]})_{21} \} .$$

Base case (2) $l - f = 0$

Apparently, $\pi_{[l]}$ is the only job with its partner undetermined. Therefore, $\pi_{[l]}$ cannot be anyone's partner except itself. Fig. 7 shows the pseudo-code for this case.

Base case (3) $l - f = 1$

In this case, only $\varpi_{[l]}$ and $\varpi_{[f]}$ are *NULL* in ϖ . $\varpi_{[l]}$ can be $\pi_{[l]}$ or $\pi_{[f]}$. Fig. 8 shows the pseudo-code for this case.

3.5 Description of the Divide-and-Conquer Method

Based on the base cases, *DC* timetabling method can be constructed as depicted in Fig. 9.

The worst case scenario happens when in every recursion, the condition of Step 9 is true, i.e., 6 half-size input instances have to be considered. Let $T(n)$ denote the worst-case running time on the input instances of size n . In each recursion, the algorithm divides the input into at most 6 pieces of size $n/2$ and finally it spends $O(n)$ time to combine the solutions from the two recursive calls. Thus the running time $T(n)$ satisfies the recurrence relation $T(n) \leq 6T(n/2) + cn$. Therefore, the algorithm is bounded by $O(n^{\log_2 6}) = O(n^{2.58})$.

4. A Timetable Enhancing Algorithm

Gilmore and Gomory converted the two-machine no-wait flow shop problem into a TSP which can be solved in $O(n \log n)$ time complexity (Tapan et al., 2006). The schedule obtained by *DC* can be mapped to a two-machine no-wait flow shop problem and further improved by the Gilmore and Gomory algorithm (*GG*).

By performing the *DC*, the best partner sequence ϖ of the given sequence π is obtained. Then, a set of super jobs can be constructed as $\{<\pi_{[1]}, \varpi_{[1]}>, \dots, <\pi_{[n]}, \varpi_{[n]}>\}$. Each super job is tranformed to a no-wait flow shop type job as in Liaw (2008). Similar to the description of the positive job and the negative job, a super job $<\pi_{[i]}, \varpi_{[i]}>$ can be described as following.

$$(1) \text{ If } s(\pi_{[i]}) = \text{True} \text{ and } s(\varpi_{[i]}) = \text{False}, \delta(<\pi_{[i]}, \varpi_{[i]}>) = \begin{pmatrix} p_{\pi_{[i]}}^1 & p_{\varpi_{[i]}}^2 \\ p_{\varpi_{[i]}}^1 & p_{\pi_{[i]}}^2 \end{pmatrix}.$$

$$(2) \text{ If } s(\pi_{[i]}) = \text{False} \text{ and } s(\varpi_{[i]}) = \text{True}, \delta(<\pi_{[i]}, \varpi_{[i]}>) = \begin{pmatrix} p_{\varpi_{[i]}}^1 & p_{\pi_{[i]}}^2 \\ p_{\pi_{[i]}}^1 & p_{\varpi_{[i]}}^2 \end{pmatrix}.$$

$$(3) \text{ If } s(\pi_{[i]}) = s(\varpi_{[i]}), \delta(<\pi_{[i]}, \varpi_{[i]}>) = \delta(\pi_{[i]}).$$

Given a discription of a super job, its corresponding no-wait flow shop type job can be decribed by a body, a head, and a tail as in Liaw (2008). The length of the body $\gamma_{[i]}$ equals to $\min\{p_{\pi_{[i]}}^1, p_{\varpi_{[i]}}^1\} + \min\{p_{\pi_{[i]}}^2, p_{\varpi_{[i]}}^2\}$. The length of head $\alpha_{[i]}$ equals to $\delta(<\pi_{[i]}, \varpi_{[i]}>)_{11} - \delta(<\pi_{[i]}, \varpi_{[i]}>)_{21}$. The length of tail $\beta_{[i]}$ equals to $\delta(<\pi_{[i]}, \varpi_{[i]}>)_{21} - \delta(<\pi_{[i]}, \varpi_{[i]}>)_{22}$. $\gamma_{[i]}$ must be positive or zero. $\alpha_{[i]}$ and $\beta_{[i]}$ can be either positive or negative. Therefore, a super job is transformed to a no-wait flow shop type job described by $\begin{pmatrix} \alpha_{[i]} & 0 \\ 0 & \beta_{[i]} \end{pmatrix}$ and an independent body length $\gamma_{[i]}$.

GG algorithm still applies to the two-machine no-wait flow shop problem with negative length operations. Thus, the optimal makespan for a certain set of super jobs can be calculated by the *GG* algorithm in the time complexity $O(n \log n)$.

The schedule obtained by the *DC* method must satisfy the secondary time order. *GG* algorithm ignores the secondary time order constraint, but constructs the

set of super jobs based on the output of *DC*. Therefore, the schedule obtained by the *DC* method can be improved by the *GG* method.

5. Proposed Greedy Search for Sequencing Problem

A simple greedy search is introduced for the sequencing problem. The pair-wise exchange function $pair_wise(\pi, i, j)$ is selected to generate a new sequence by exchanging $\pi_{[i]}$ and $\pi_{[j]}$. In each iteration, $\pi_{[i]}$ is exchanged with each $\pi_{[j]}$ ($j > i$) to generate a new sequence and the new sequence will be explored by the *DC* and *GG* algorithms. Other neighbor generating methods such as one-insertion can also be applied to generate new sequences to explore. The search will stop when no improvement has been made. Because the algorithm stops when no better solution can be found, it is a greedy search. Experimental results show that the initial processing sequence has no effect on the final result, therefore, an initial processing sequence is generated randomly in this paper. The procedure of the proposed greedy search is shown in Fig. 10.

Since the number of iterations cannot be determined, the time complexity for the proposed search is unknown.

To illustrate the computational procedure of the proposed greedy search algorithm, a 10-jobs 2-machines instance as shown in Table 2 is considered. An initial processing sequence randomly obtained is (3, 0, 5, 1, 7, 2, 4, 8, 9, 6) with makespan 249. Eight iterations are performed when the greedy search stops; the corresponding results are shown in Table 3.

6. Computational Experiments

The proposed greedy algorithm (*GS*) is coded in Java and implemented on an Intel Core 2 Duo 2.2 GHz personal computer. Test problems with size n equal to 40, 80, 120, 160 and 200 are randomly generated. For each operation, an integer processing time is generated from the uniform distribution $U[1,50]$. For each problem size, 100 problem instances are generated.

In order to evaluate the performance of the proposed approach, the solution obtained is compared with a lower bound of the problem and the results obtained by the algorithm presented in Liaw (2008). The lower bound adopted is based on the optimal solution of a relaxed problem $J_2 // C_{\max}$ which was optimally solved in Jackson (1956). The algorithm presented in Liaw (2008) adopted the *GG* approach as the timetabling method and Tabu search for the sequencing problem.

Table 4 shows the average time consumption of *DC* and *DC_GG*. For each problem size, 10 test problems are generated and the proposed timetabling method performs on 100 randomly generated processing sequences for each test problem. $T_{avg}(ms)$ is the average CPU time in milliseconds for running *DC* on one processing sequence.

Table 4 indicates that the CPU time required by the proposed timetabling method increases with the problem size. Although *DC* can find the best timetable in polynomial complexity, its exponent is approximately 2.58, which contributes to the high time requirement for large size problems.

Table 5 shows the detailed results for the proposed greedy search algorithm. $D_{avg}(\%)$ is the average percentage deviation from the lower bound of the solution obtained by the algorithm described in Jackson (1956) and $D_{\max}(\%)$ is the maximal percentage deviation from the lower bound of the solution. N_{opt} is the number of problem instances (out of N , N is set to 100 in this paper) for which the solution value obtained by the corresponding algorithm equals to the lower bound. Apparently, these solutions are optimal. $T_{avg}(s)$ is the average CPU time in second and $T_{\max}(s)$ is the maximum CPU time in second. *GG_TS* is the algorithm proposed in Liaw (2008).

In order to show how much *DC* and *GS* contribute with respect to solution quality, the results obtained by *GG_GS* are compared to the other two algorithms.

As shown in Table 5, the proposed algorithm can find 96% optimal solutions on average, while *GG_TS* and *GG_GS* can find 89% and 91% optimal solutions on average, respectively. D_{avg} s of the proposed algorithm are better than these of *GG_TS* and *GG_GS* on all instance groups. D_{max} s of the proposed algorithm are better than these of *GG_TS* except the instances of size 40 and size 80. D_{max} s of the proposed algorithm are better than *GG_GS* on all instance groups. *GG_GS* requires a little more computation time than *GG_TS*, while the proposed algorithm requires more than 11 times of computation time than *GG_TS* at most.

The only difference between *GG_GS* and *GG_TS* is that they adopt different searching methods for the sequencing problem. Based on the comparison of these two algorithms, *GS* outperforms *TS* with respect to solution quality. The proposed algorithm uses *DC* and *GG* as its timetabling method which is more complicated than that of *GG_GS*. Because the complexity of *DC* is $O(n^{2.58})$, calculating a timetable consumes more CPU time with the increase of the problem size n . Therefore, the CPU time required by the proposed algorithm increases greatly with the increase of the problem size. However, it is affordable with respect to its effectiveness.

7. Conclusions

In this paper, a greedy search algorithm integrated with a divide-and-conquer-based timetabling method is proposed for the two-machine no-wait job shop problem with makespan minimization. The generic framework of the existing algorithms for the considered problem has been adopted by decomposing a problem into the timetabling problem and the sequencing problem, which were solved by the proposed divide-and-conquer-based timetabling method and a greedy search, respectively. The

proposed divide-and-conquer method (*DC*) can solve the timetabling problem in $O(n^{2.58})$ complexity for a given processing sequence. Based on the results obtained by *DC*, a timetable enhancing algorithm converted the timetabling problem to the two-machine flow shop problem which can be optimally solved by the well-known Gilmore and Gomory algorithm (*GG*). A simple greedy search (*GS*) embedded with *DC* and *GG* is proposed for the sequencing problem. Due to the complexity of the timetabling method, calculating a timetable consumes more CPU time with the increase of the problem size n . Therefore, the CPU time required by *GS* increases greatly with the increase of the problem size. Experimental results showed that the proposed *GS* can find 96% optimal solutions on average. With respect to the effectiveness of the proposed *GS*, it is affordable.

Acknowledgements

This work is supported in part by the National Natural Science Foundation of China under Grants 60973073 and 61070160, Program for New Century Excellent Talents in University (NCET-09-0290), Qing Lan Project, and the Scientific Research Foundation of Graduate School of Southeast University (No. YBJJ0931).

References

- Amir M, Zandieh M, Yazdani M, Bagheri A, 2010. A variable neighbourhood search algorithm for the flexible job-shop scheduling problem. *International Journal of Production Research* 48(19), 5671-5689.
- Bansal N, Mahdian M, Sviridenko M (2005) Minimizing Makespan in No-Wait Job Shops. *Mathematics of Operations Research* 30 (4): 817–831.
- Blazewicz J, Domschke W, Pesch E (1996) The job-shop-scheduling-problem: conventional and new solution techniques. *European Journal of Operational Research* 93: 1–33.
- Dinh-Nguyen P, Andreas K (2008) Surgical case scheduling as a generalized job shop scheduling problem. *European Journal of Operational Research* 185: 1011–1025.
- Framinan JM, Schuster C (2006) An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach. *Computers & Operations Research* 33(1): 1200–13.
- Gilmore PC, Gomory RE (1964) Sequencing a one state-variable machine: a solvable case of the traveling salesman problem. *Operations Research* 12: 655-679.
- Grabowski J, Pempera J (2000) Sequencing of jobs in some production system. *European Journal of Operational Research* 125: 535–550.

- Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG (1979) Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics* 5:287-326.
- Hall NG, Sriskandarajah C (1996) A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research* 44: 510–525.
- Jackson JR (1956) An extension of Johnson's results on job lot scheduling. *Naval Research Logistics Quarterly* 3:201-203.
- Jain A, Meeran S (1990) Deterministic job-shop-scheduling: past, present and future. *European Journal of Operational Research* 113 (2): 390–434.
- Lenstra JK, Rinnooy Kan AHG (1979) Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics* 4: 121–140.
- Liaw CF (2008) An efficient simple metaheuristic for minimizing the makespan in two-machine no-wait job shops. *Computers & Operations Research* 35: 3276-3283.
- Macchiaroli R, Mole S, Riemma S (1999) Modelling and optimization of industrial manufacturing processes subject to no-wait constraints. *International Journal of Production Research* 37(11): 2585–2607.
- Mascis A, Pacciarelli D (2002) Job shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research* 142(3): 498–517.
- Meloni C, Pacciarelli D, Pranzo M (2004) A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research* 131: 215–235.
- Ovacik I, Uzsoy R (1997) Decomposition method for complex factory scheduling problems. Dordrecht: Kluwer Academic Publishing.
- Raaymakers W, Hoogeveen J (2000) Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research* 126: 131–151.
- Rajendran C (1994) A no-wait flowshop scheduling heuristic to minimize makespan. *Journal of the Operational Research Society* 45(4): 472–478.
- Rajkumar M., Asokan P., Vamsikrishna V., 2010. A GRASP algorithm for flexible job-shop scheduling with maintenance constraints. *International Journal of Production Research* 48(22), 6821-6836.
- Reddi S, Ramamoorthy C (1973) A scheduling-problem. *Operational Research Quarterly* 24: 441–446.
- Sahni S, Cho Y (1979) Complexity of scheduling shops with no-wait in process. *Mathematics of Operations Research* 4: 448–457.
- Schuster C, Framinan JM (2003) Approximate procedures for no-wait job shop scheduling. *Operations Research Letters* 31:308–318.
- Schuster C (2006) No-wait job shop scheduling: Tabu search and complexity of subproblems. *Mathematical Methods of Operations Research* 63: 473-491.
- Tapan PB, Jatinder NDG, Chelliah S (2006) A review of TSP based approaches for flowshop scheduling. *European Journal of Operational Research* 169: 816-854.
- Wismar DA (1972) Solution of the flowshop scheduling-problem with no intermediate queues. *Operations Research*; 20: 689–697.
- Woeginger G.J., 2004. Inapproximability results for no-wait job shop scheduling. *Operations Research Letters* 32: 320-325.
- Zandieh M, Adibi MA., 2010. Dynamic job shop scheduling using variable neighbourhood search. *International Journal of Production Research* 48(8), 2449-2458.

Zhu J, Li XP, Wang Q (2009) Complete local search with limited memory algorithm for no-wait job shops to minimize makespan. European Journal of Operational Research 198(2):378-386.

Table 1. Notations for two-machine no-wait job shop problems

$n \in N$	: Number of jobs
$J = \{J_1, J_2, \dots, J_n\}$	: Set of jobs
$M = \{M_1, M_2\}$	: Set of machines
$o_{\pi[i]}^k, k=1,2$	: The k th operation of $\pi_{[i]}$
$p_{\pi[i]}^k, k=1,2$	: The duration of $o_{\pi[i]}^k$
$t_{\pi[i]} \in N$	: Starting time of $\pi_{[i]}$
$l_{\pi[i]} = p_{\pi[i]}^1 + p_{\pi[i]}^2$	: Duration of $\pi_{[i]}$
$T_\pi = (t_{\pi[1]}, \dots, t_{\pi[n]})$	: Timetable of π
$t_{\pi[i]}^k \in N, k=1,2$	: Starting time of $\pi_{[i]}$ on M_k

Table 2. A 10-jobs 2-machines instance

i	p_i^1	p_i^2	$s(i)$
0	11	13	<i>Ture</i>
1	21	33	<i>False</i>
2	35	8	<i>False</i>
3	20	21	<i>False</i>
4	37	31	<i>Ture</i>
5	14	2	<i>Ture</i>
6	15	45	<i>Ture</i>
7	10	12	<i>Ture</i>
8	44	13	<i>Ture</i>
9	4	38	<i>False</i>

Table 3. Solutions for the instance

No.	Result sequence	Makespan
1	(4, 2, 8, 0, 5, 9, 3, 7, 1, 6)	248
2	(0, 6, 4, 8, 3, 9, 1, 7, 2, 5)	246
3	(4, 9, 0, 3, 7, 1, 2, 6, 8, 5)	238
4	(7, 1, 8, 0, 2, 9, 6, 3, 4, 5)	237
5	(3, 6, 9, 8, 0, 2, 5, 7, 1, 4)	236
6	(7, 6, 8, 9, 4, 1, 0, 3, 2, 5)	235
7	(6, 9, 7, 3, 4, 0, 1, 8, 5, 2)	232
8	(6, 9, 0, 5, 2, 4, 1, 8, 7, 3)	231

Table 4The time consuming of the divide and conquer timetabling method

n	$T_{avg} (ms)$	
	DC	DC_GG
40	0.47	1.25
80	1.26	3.43
120	3.72	6.08
160	6.25	9.67

Table 5 Computational results of GS

n	GG_TS				GG_GS				Proposed			
	D_{avg}	D_{max}	N_{opt}/N	T_{avg}	D_{avg}	D_{max}	N_{opt}/N	T_{avg}	D_{avg}	D_{max}	N_{opt}/N	T_{avg}
	(%)	(%)		(s)	(%)	(%)		(s)	(%)	(%)		(s)
40	0.09	0.694	0.8	0.064	0.03	0.87	0.88	0.49	0.022	0.567	0.91	0.73
80	0.017	0.287	0.93	0.34	0.015	0.29	0.88	1.46	0.014	0.20	0.89	6.18
120	0.003	0.033	0.9	1.4	0.001	0.03	0.97	3.30	0.0003	0.031	0.99	22.8
160	0.004	0.051	0.9	4.67	0.002	10.024	0.91	8.33	0	0	1	68.82
200	0.003	0.057	0.9	10.45	0.001	0.02	0.92	13.46	0.0002	0.019	0.99	147.12
Avg.	0.023	0.224	0.89		0.01	0.25	0.91		0.007	0.167	0.96

Fig. 1 A positive job and a negative job

Fig. 2 Gantt charts of the feasible timetable for possible two jobs

Fig. 3. Profile of Gantt chart for two subsequences when $\varpi[k] = \pi[k]$

Fig. 4. Profile of Gantt chart for two subsequences when $\varpi[k] = \pi[k-1]$

```

/*Base case (1)*/
Function  $OPT\_B1(\pi, \varpi)$ 
1.  $sum \leftarrow 0$ ,  $overlap \leftarrow 0$ 
/*Calculating the sum of all super jobs*/
2. For  $i = 1$  to  $|\pi|$ 
    /*If  $\varpi[i] = \pi[i-1]$ , the super job has been visited*/
    If  $\varpi[i] \neq \pi[i-1]$ ,
         $sum \leftarrow sum + C_{\max}(<\pi[i], \varpi[i]>)$ 
/*Calculating the sum of overlaps
between any two consecutive super jobs*/
3. For  $i = 1$  to  $|\pi| - 1$ 
    3.1 If  $\varpi[i] = \pi[i-1]$ ,  $i \leftarrow i + 1$ 
    3.2  $j \leftarrow i + 1$ 
    3.3 If  $\varpi[j] = \pi[j]$ ,  $j \leftarrow j + 1$ 
    3.4
     $overlap \leftarrow overlap + Overlap(<\pi[i], \varpi[i]>, <\pi[j], \varpi[j]>)$ 
4. Return  $sum - overlap$ 

/*Calculating the duration of a super job*/
Function  $C_{\max}(<\pi[i], \varpi[i]>)$ 
Return  $\max\{p_{\pi[i]}^1, p_{\varpi[i]}^1\} + \max\{p_{\pi[i]}^2, p_{\varpi[i]}^2\}$ 

/*Calculating the maximal overlap of two super jobs*/
Function  $Overlap(<\pi[i], \varpi[i]>, <\pi[j], \varpi[j]>)$ 
1.  $\lambda \leftarrow \max\{p_{\pi[i]}^2, p_{\varpi[i]}^2\} + \max\{p_{\pi[j]}^1, p_{\varpi[j]}^1\}$ ;
2.  $\rho_1 \leftarrow \max\{\delta(\pi[i])_{12}, \delta(\varpi[i])_{12}\} + \max\{\delta(\pi[j])_{11}, \delta(\varpi[j])_{11}\}$ 
3.  $\rho_2 \leftarrow \max\{\delta(\pi[i])_{22}, \delta(\varpi[i])_{22}\} + \max\{\delta(\pi[j])_{21}, \delta(\varpi[j])_{21}\}$ 
4.  $\rho \leftarrow \max\{\rho_1, \rho_2\}$ 
5. Return  $\lambda - \rho$ 

```

Fig. 5. The pseudocode for Base case (1)

Fig. 6. Maximal overlap of two consecutive super jobs

```

/*Base case (2)*/
Function  $OPT\_B2(\pi, \varpi, l)$ 
1.  $\varpi_{[l]} \leftarrow \pi_{[l]}$ 
/*At this point, all jobs' partners
in the current sequence are
determined*/
2.  $c_{\max} \leftarrow OPT\_B1(\pi, \varpi)$ 
2. Return  $c_{\max}$ 

```

Fig.7. The pseudocode for Base case (2)

```

/*Base case (3)*/
Function  $OPT\_B3(\pi, \varpi, l, f)$ 
1.  $c_{\max} \leftarrow 0$ 
2. If  $s(\pi_{[l]}) = s(\pi_{[f]})$ 
2.1  $\varpi_{[l]} \leftarrow \pi_{[l]}, \varpi_{[f]} \leftarrow \pi_{[f]}$ 
2.2  $c_{\max} \leftarrow OPT\_B1(\pi, \varpi)$ 
3. If  $s(\pi_{[l]}) \neq s(\pi_{[f]})$ 
3.1  $\varpi_{[l]} \leftarrow \pi_{[f]}, \varpi_{[f]} \leftarrow \pi_{[l]}, c_1 \leftarrow OPT\_B1(\pi, \varpi)$ 
3.2  $\varpi_{[l]} \leftarrow \pi_{[l]}, \varpi_{[f]} \leftarrow \pi_{[f]}, c_2 \leftarrow OPT\_B1(\pi, \varpi)$ 
3.3  $c_{\max} \leftarrow \min\{c_1, c_2\}$ 
4. Return  $c_{\max}$ 

```

Fig.8. The pseudocode for Base case (3)

```

Function  $OPT(\pi, \varpi)$ 

```

```

/*Find indices of the first and the last NULL item in  $\varpi$ */

```

1 If $\forall \varpi_{[i]} \neq NULL (i = 1, \dots, |\pi|), l \leftarrow -1, f \leftarrow -1;$

Else $l \leftarrow \max\{i \mid \varpi_{[i]} \neq NULL, i = 1, \dots, |\pi|\}, f \leftarrow \min\{i \mid \varpi_{[i]} \neq NULL, i = 1, \dots, |\pi|\};$

/*Step 2~4 deal with the base cases*/

2. If $l = f = -1$, return $OPT_B1(\pi, \varpi);$

3. If $l - f = 0$, return $OPT_B2(\pi, \varpi, l);$

4. If $l - f = 1$, return $OPT_B3(\pi, \varpi, l, f);$

/* k is the index of the key job for the division*/

5. $k \leftarrow \left\lfloor \frac{l+f}{2} \right\rfloor$

/* DC_Case (1) */

6. If $s(\pi_{[k-1]}) = s(\pi_{[k]}) = s(\pi_{[k+1]})$, $\varpi_{[k]} \leftarrow \pi_{[k]}, c_{\max} \leftarrow$
 $OPT(\pi(1, k), \varpi(1, k)) + OPT(\pi(k, n), \varpi(k, n))$
 $-(\max\{p_{\pi[k]}^1, p_{\varpi[k]}^1\} + \max\{p_{\pi[k]}^2, p_{\varpi[k]}^2\})$

/* DC_Case (2) */

7. If $s(\pi_{[k-1]}) = s(\pi_{[k]})$ and $s(\pi_{[k]}) \neq s(\pi_{[k+1]})$

7.1 $\varpi_{[k]} \leftarrow \pi_{[k]};$

$c_1 \leftarrow OPT(\pi(1, k), \varpi(1, k)) + OPT(\pi(k, n), \varpi(k, n))$
 $-(\max\{p_{\pi[k]}^1, p_{\varpi[k]}^1\} + \max\{p_{\pi[k]}^2, p_{\varpi[k]}^2\});$

7.2 $\varpi_{[k]} \leftarrow \pi_{[k+1]}, \varpi_{[k+1]} \leftarrow \pi_{[k]};$

$c_2 \leftarrow OPT(\pi(1, k+1), \varpi(1, k+1)) + OPT(\pi(k, n), \varpi(k, n))$
 $-(\max\{p_{\pi[k+1]}^1, p_{\varpi[k+1]}^1\} + \max\{p_{\pi[k+1]}^2, p_{\varpi[k+1]}^2\});$

7.3 $c_{\max} \leftarrow \min\{c_1, c_2\};$

7.4 If $c_{\max} = c_1$, $\varpi_{[k]} \leftarrow \pi_{[k]};$ else $\varpi_{[k]} \leftarrow \pi_{[k+1]}, \varpi_{[k+1]} = \pi_{[k]};$

/* DC_Case (3) */

8. If $s(\pi_{[i-1]}) \neq s(\pi_{[i]})$, $s(\pi_{[i]}) = s(\pi_{[i+1]})$

8.1 $\varpi_{[k]} \leftarrow \pi_{[k]};$

$c_1 \leftarrow OPT(\pi(1, k), \varpi(1, k)) + OPT(\pi(k, n), \varpi(k, n))$
 $-(\max\{p_{\pi[k]}^1, p_{\varpi[k]}^1\} + \max\{p_{\pi[k]}^2, p_{\varpi[k]}^2\});$

8.2 $\varpi_{[k]} \leftarrow \pi_{[k-1]}, \varpi_{[k-1]} \leftarrow \pi_{[k]};$

$$c_2 \leftarrow OPT(\pi(1, k), \varpi(1, k)) + OPT(\pi(k-1, n), \varpi(k-1, n)) \\ - (\max\{p_{\pi[k]}^1, p_{\varpi[k]}^1\} + \max\{p_{\pi[k]}^2, p_{\varpi[k]}^2\});$$

8.3 $c_{\max} \leftarrow \min\{c_1, c_2\};$

8.4 If $c_{\max} = c_1$, $\varpi_{[k]} \leftarrow \pi_{[k]};$ else $\varpi_{[k]} \leftarrow \pi_{[k-1]}, \varpi_{[k-1]} = \pi_{[k]};$

/* DC_Case (4) */

9. If $s(\pi_{[i-1]}) = s(\pi_{[i+1]}), s(\pi_{[i]}) \neq s(\pi_{[i+1]})$

9.1 $\varpi_{[k]} \leftarrow \pi_{[k]};$

$$c_1 \leftarrow OPT(\pi(1, k), \varpi(1, k)) + OPT(\pi(k, n), \varpi(k, n)) \\ - (\max\{p_{\pi[k]}^1, p_{\varpi[k]}^1\} + \max\{p_{\pi[k]}^2, p_{\varpi[k]}^2\});$$

9.2 $\varpi_{[k]} \leftarrow \pi_{[k-1]}, \varpi_{[k-1]} \leftarrow \pi_{[k]};$

$$c_2 \leftarrow OPT(\pi(1, k), \varpi(1, k)) + OPT(\pi(k-1, n), \varpi(k-1, n)) \\ - (\max\{p_{\pi[k]}^1, p_{\varpi[k]}^1\} + \max\{p_{\pi[k]}^2, p_{\varpi[k]}^2\});$$

9.3 $\varpi_{[k]} \leftarrow \pi_{[k+1]}, \varpi_{[k+1]} \leftarrow \pi_{[k]};$

$$c_3 \leftarrow c_3 + OPT(\pi(1, k+1), \varpi(1, k+1)) + OPT(\pi(k, n), \varpi(k, n)) \\ - (\max\{p_{\pi[k+1]}^1, p_{\varpi[k+1]}^1\} + \max\{p_{\pi[k+1]}^2, p_{\varpi[k+1]}^2\});$$

9.4 $c_{\max} \leftarrow \min\{c_1, c_2, c_3\}$

9.5 If $c_{\max} = c_1$, $\varpi_{[k]} \leftarrow \pi_{[k]};$

Else if $c_{\max} = c_2$ $\varpi_{[k]} \leftarrow \pi_{[k-1]}, \varpi_{[k-1]} = \pi_{[k]};$

Else $\varpi_{[k]} \leftarrow \pi_{[k+1]}, \varpi_{[k+1]} \leftarrow \pi_{[k]};$

10. Return $c_{\max}$

Procedure *Divide_and_Conquer*(π)

1. $\varpi \leftarrow (NULL, \dots, NULL), c_{\max} \leftarrow 0, n \leftarrow |\pi|$

$$T(t_{\pi[1]}, \dots, t_{\pi[n]}) \leftarrow (0, -1, \dots, -1)$$

2. $c_{\max} \leftarrow OPT(\pi, \varpi)$

3. For $i = 2$ to n

3.1 If $\varpi_{[i]} = \pi_{[i]}, t_{\pi[i]} \leftarrow \max\{t_{\pi[i-1]} + l_{\pi[i-1]}, t_{\varpi[i-1]} + l_{\varpi[i-1]}\}$

Fig. 9. The pseudo-code of *DC* timetabling method

Fig.10. The procedure of the proposed *GS*