Computing and Informatics, Vol. 24, 2005, 297–319

PRODUCTION SCHEDULING WITH COMPLEX PRECEDENCE CONSTRAINTS IN PARALLEL MACHINES

Katerina El Raheb, Christos T. Kiranoudis

National Technical University of Athens
School of Chemical Engineering
Department of Process Analysis & Plant Design
Zografou Campus, 15780, Athens, Greece

Panagiotis P. Repoussis, Christos D. Tarantilis

Athens University of Economics & Business
Department of Management Science & Technology
Management Science Laboratory
Evelpidon 47A & Leukados 33, 11362, Athens, Greece
e-mail: tarantil@aueb.gr

Manuscript received 24 June 2004; revised 17 May 2005
Communicated by Richard Wait

Abstract. Heuristic search is a core area of artificial intelligence and the employment of an efficient search algorithm is critical to the performance of an intelligent system. This paper addresses a production scheduling problem with complex precedence constraints in an identical parallel machines environment. Although this particular problem can be found in several production and other scheduling applications; it is considered to be $\mathcal{NP}$-hard due to its high computational complexity. The solution approach we adopt is based on a comparison among several dispatching rules combined with a diagram analysis methodology. Computational results on large instances provide relatively high quality practical solutions in very short computational times, indicating the applicability of the methodology in real life production scheduling applications.

Keywords: Heuristic search, precedence constraints, parallel machine scheduling

1 INTRODUCTION

Parallel machine scheduling has been a source of challenging problems for researchers in the area of computer and manufacturing engineering. The general problem of parallel machine scheduling can be stated as scheduling a set of partially ordered jobs (or computational tasks) onto a parallel machine (or multiprocessor) system so that a set of performance criteria will be optimized. The difficulty of the problem depends heavily on the topology of the job (task) graph representing the precedence relations among the jobs, the topology and the number of parallel machines, the uniformity of each job processing time, and the performance criteria chosen [1].

The parallel machine scheduling problem (PMSP) is computationally intractable even under simplified assumptions [2]. In particular, the problem restricted to two machines ($P2||C_{max}$) was shown in the ordinary sense to be $\mathcal{NP}$-hard by a bi-partitioning problem, as stated by Karp [3]. With precedence constraints the PMSP is as hard as the classical problem of scheduling precedence constraint Unit Execution Time tasks (UET) on parallel machines, as a task moldable to run with the same execution time on any number of machines. Therefore, because of this computational complexity issue, heuristic algorithms have been proposed to obtain optimal and sub-optimal solutions.

Recently, the parallel machine environment scheduling received a lot of attention due to its practical use. In particular, Bilge et al. [4], Timkovsky [5], Lepere et al. [6] and Hurink and Knust [7] considered the parallel machine scheduling problem as Malleable Tasks (MT), allowing each task to be executed in separate processors. Moreover, parallel task scheduling is one of the most important problems in parallel computation. Bampis et al. [8] propose a $(2 - 2/(2m + 1))$-approximation heuristic algorithm for the $Pm|prec;\ c_{ij} = 1;\ p_i > 1|C_{max}$ using a directed acyclic graph representation where the vertices represent the tasks to be executed and the arcs correspond to the communication delays. The parallel architecture is composed by a set of identical processors and the objective is to find a feasible schedule minimizing the *makespan*, i.e. the time at which the last task of the graph finishes its execution.

Due to the importance of scheduling problems, there is a vast literature that addresses modelling and solution aspects of several instances of PMPS problems, including either exact (complete) or approximate algorithms. Exact algorithms guarantee to find for every finite size instance an optimal solution, in bounded time. Among various branch & bound and dynamic programming algorithms proposed for PMPSs, the approach adopted by Belouadah and Potts [9] in which lower bounds are obtained by a Lagrangian relaxation of machine capacity constraints appears best; $P||\sum w_j C_j$ problems with up to 20 jobs and 8 machines can be solved within reasonable time. Contrary to exact approaches, approximate methodologies does not guarantee of finding optimal solutions for the sake of getting good solutions in significantly reduced amount of time. Among the basic approximate methods they are usually distinguished between construction and local search heuristics. Construction heuristics generate solutions from scratch by adding-to an initial empty partial constructed solution-components, until a feasible solution is complete. Such heuristics typically produce mediocre-to-good solutions in relatively short computational times.

Local search heuristics start from some initial solution and iteratively try to replace the current solution with a "better" solution in an approximately defined neighborhood of the current solution. Based on this concept, another class of iterative improvement algorithms, called metaheuristics, has emerged which basically tries to combine basic heuristic in higher level frameworks, resulting a more efficient and effective exploration of the search space. Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase its performance (higher quality solutions) but in relatively large computational times. Their main goal is to avoid the disadvantages of iterative improvement and, in particular, through multiple descents by the local (neighborhood) search to escape from local optima. Sophisticated metaheuristics, such as genetic algorithms, tabu search and simulated annealing, are only some of the improvement methods proposed, able to find near optimal solutions for several scheduling problems [1, 10, 11, 12].

However, there are certain instances of PMPS problems that constructive heuristics may provide near-optimal solutions [13]. Recently, Dunstall and Wirth [14] considered an identical parallel machine-scheduling problem, within which jobs arranged into families and sequence-independent setup time between jobs of different families on these machines. They evaluate the performance of previously appeared in literature heuristics, relative to lower bounds and solutions obtained using an exact algorithm. Moreover, Gendreau et al. [15] propose a Divide and Merge heuristic and several lower bounds for the $P|s_{ij}|C_{max}$ problem. In particular, they compare their heuristic with a Tabu Search algorithm proposed earlier by França et al. [16] and the results yield similar quality solutions but with significantly reduced amount of running time.

Evidently, there are certain instances of PMPS problems and practical applications where hundreds of jobs/operations have to be determined in minutes, in which low-level heuristic methods have been shown to yield high-quality practical solutions in much shorter development time than that of other approaches. It is precisely in this context that hyperheuristic approaches have been proposed as heuristics that operate at a higher level of abstraction than current metaheuristic approaches [17]. A hyperheuristic is a high-level heuristic that adaptively chooses between several low-level knowledge-poor heuristics so that while using only cheap, easy-to-implement low-level heuristics, it may achieve to produce solution quality approaching that of expensive knowledge-rich approaches (tabu search and genetic algorithms), in a reasonable amount of CPU time. At each decision point the hyperheuristic must choose which low-level heuristic to apply, without recourse to any domain knowledge. Such hyperheuristics have been successfully applied by Cowling et al. [17] to a real-world problem of personnel scheduling problem.

In this paper we present an efficient heuristic method to address a parallel machine production scheduling problem, with multiple product types, each with complex precedence constraints, large number of jobs/production phases and non uniform processing times. It is worth mentioning that parallel machine scheduling with multiple product types and complex precedence constraints has not, to our knowledge, been addressed before. Our solution procedure is "intuitively intelligent" since it examines the performance of several dispatching rule applications combined with a diagram analysis methodology. In particular, the solution approach is divided into two phases as follows:

a) Loading the required jobs and developing a sequence between them through the application of heuristic nature dispatching rules, and

b) Timing the jobs in the machines by utilizing the Critical Path Method [18].

The usage of dispatching rules plays a two-fold role. Firstly, several different dispatching rules are used to allocate the jobs to machines and to create the sequence with respect to the precedence constraints. Secondly, the initial scheduling is "improved" by selecting the dispatching rule performed best in each case.

The proposed methodology, in contrary to classical metaheuristic, has the advantage of solving real-life production scheduling problems, with relatively large number of jobs and machines, in adequately short computational time. Furthermore, low level heuristic search is a major component of several intelligent systems, and central research subject of applied artificial intelligence. As the field of artificial intelligence moves toward building complex intelligent systems to tackle real world problems in various domains [19, 20, 21], the need of employing efficient and effective heuristic algorithms becomes increasingly important.

The remainder of the paper is organized as follows: First we present a comprehensive description and formulation of the parallel machine scheduling problem with complex precedence constraints, as applied to the production of several different types of products. Then, we describe the attributes and the mechanisms of the proposed methodology. Subsequently, we provide a thorough analysis of the computational results obtained by applying our methodology on three different large-scale problems. In particular, we seek to study the behavior of several dispatching rules and uncover relevant properties of the scheduling problem considered. Finally, we conclude the paper and offer pointers for future research.

2 PROBLEM DESCRIPTION AND FORMULATION

The parallel machine scheduling problem considered in this paper is based on the deterministic model; that is, the execution time and the relationship between jobs are known. The precedence relationship among the jobs is represented by an acyclic directed graph, and job execution time can be nonuniform. We assume that the parallel machine system is uniform and nonpreemptive; that is, the machines are identical and complete the current job before executing a new one.

Using the three field notation of [13] this instance of production scheduling with complex precedence constraints in parallel machines can be formulated as a deterministic $Pm|prec|C_{max}$ problem which in our case is stated as follows: Find the scheduling operations of m identical parallel machines $M = \{1, 2, \ldots, m\}$ processing $K = \{1, 2, \ldots, k\}$ types of products $P = \{P_1, P_2, \ldots, P_K\}$. In particular, G_K items of each product type is produced in Φ_K phases with specific precedence constraints, for each product type k produced, respectively. Thus, the total number of produced products pp is equal to the sum of products produced of each type k, $pp = \sum_{K=1}^{k} G_K$. Similarly, the total number of jobs n to be scheduled is equal to the sum of products, the number of phases Φ_K times the number of products G_K, to be produced for each type k, $n = \sum_{K=1}^{k}(G_K \Phi_K)$. Each phase of production is considered as a job denoted as $J^g_{\phi K}$, where $g = 1, 2, \ldots, G_K$, $\phi = 1, 2, \ldots, \Phi_K$, and $K = 1, 2, \ldots, k$. Therefore, a set N of n independent jobs $N = \{J^1_{11}, J^1_{21}, \ldots, J^1_{\Phi_1 1}, J^2_{11}, J^2_{21}, \ldots, J^2_{\Phi_1 1}, \ldots, J^{G_1}_{\Phi_1 1}, \ldots, J^{G_2}_{\Phi_2 2}, \ldots, J^{G_K}_{\Phi_K k}\}$ must be scheduled on m identical parallel machines. Each job $i \in N$ has m processing times p_{ij}, where $j \in M$, a weight w_i, a due date d_i and other problem dependent parameters.

Generally, p_{ij} is a predefined characteristic of each job and represents the actual processing time of job i if it is processed on machine j including setup time for every job in each machine. In the case of identical machines, all machines have the same speed and hence processing times of a job are identical on different machines, i.e. $p_{ij} \equiv p_i$. Moreover, all jobs are independent from the job sequence and the machine to be processed, all jobs are available for processing at time zero (assuming precedence feasibility) and no preemption is allowed during processing. The objective is to find a schedule, i.e. an allocation of each job to a time interval on one machine, such that the completion time (*makespan*) is minimized (the *makespan* is denoted by C_{max}). Minimization of *makespan* is expected to maximize the total throughput and to minimize machine's idle times and the number of slack jobs.

As stated above, there exists a predetermined job ordering restriction that specifies, for each job i, a set of jobs that must be scheduled before or after job i. To explicitly take into account the possible job ordering restriction we only need to consider those schedules where the partial schedule on each machine is feasible. Thus, the following sets, for $i \in N$ and $z \in M$, are defined:

$$A^z_j = \{i \in N \mid i \text{ can succeed } j \text{ in a feasible partial schedule on machine } z\}$$

$$B^z_j = \{i \in N \mid i \text{ can precede } j \text{ in a feasible partial schedule on machine } z\}.$$

The mathematical programming formulation of the $Pm|prec|C_{max}$ requires three groups of 0-1 variables to model the sequence in which jobs are assigned to each machine and are defined as follows:

$$x^z_{ij} = \begin{cases} 1 & \text{if job } j \text{ is processed immediately after job } i \\ & \text{in the sequence of jobs scheduled on machine } z \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$x^z_{0j} = \begin{cases} 1 & \text{if job } j \text{ is processed first on machine } z \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$x^z_{j,n+1} = \begin{cases} 1 & \text{if job } j \text{ is processed last on machine } z \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

Let C_j denote the completion time of job j in a schedule. Given the above-defined variables the problem can be formulated as follows:

$$\min \left(\max_{j \in N} C_j \right) \tag{4}$$

Subject to

$$\sum_{z \in M} \sum_{i \in B^z_j \cup \{0\}} x^z_{ij} = 1, \qquad \forall j \in N \tag{5}$$

$$\sum_{j \in N} x^z_{0j} \leq 1, \qquad \forall z \in M \tag{6}$$

$$\sum_{i \in B^z_j \cup \{0\}} x^z_{ij} = \sum_{i \in A^z_j \cup \{n+1\}} x^z_{ji}, \qquad \forall j \in N, \forall z \in M \tag{7}$$

$$C_j = \sum_{z \in M} \left(p_{jz} x^z_{0j} + \sum_{i \in B^z_j} (C_i + p_{jz})\, x^z_{ij} \right), \qquad \forall j \in N \tag{8}$$

$$x^z_{ij} \in \{0,1\}, \qquad \forall i, j \in N, z \in M. \tag{9}$$

The objective function (4) seeks to minimize the *makespan* C_{max}. Constraint (5) ensures that each job is assigned to one and only one machine. Constraint (6) ensures that each machine is utilized at most once. Constraint (7) guarantees that the assignment of jobs to machines is well defined. Equality constraint (8) defines completion time C_j. The last constraint (9) represents binary integrality requirement of 0-1 variables. Constraints (7), (8) and (9) ensures that schedule on each machine is feasible according to the explicitly defined precedence constraints. However, since all machines are identical, we do not need to distinguish different machines, and hence the formulation (4–9) can be simplified. Moreover, a problem without precedence constraints, if we do not know any ordering pattern that an optimal schedule must follow, thus any schedule is feasible and simply $A^z_j = B^z_j = N \backslash \{j\}$ for all j and z.

The model of (4–9) is a typical binary program that mathematically depicts the parallel machine production scheduling problem with precedence constraint. It is a complex combinatorial optimization model that requires substantial effort for determining approximate solutions even for medium size problems. Due to the computational complexity problems, heuristic approaches are appropriate for obtaining solutions of relatively good quality in reasonable computational times. In order to evaluate the worst-case performance of an approximate algorithm, we recall the definition of the *relative performance* of a heuristic h:

$$\rho^h = \max_G \frac{C^h_{max}(G)}{C^{opt}_{max}(G)} \tag{10}$$

where $C^{opt}_{max}(G)$ denotes the optimal *makespan* of a feasible schedule of a graph G, and $C^h_{max}(G)$ the *makespan* obtained by the heuristic h [8]. These performance ratios are either constant or may depend on some instance input data like the number of machines, tasks or precedence relation. However, optimal solutions can only be achieved by implicit enumerations schemes and in many cases the optimal values could not be computed in reasonable time unless $\mathcal{P} = \mathcal{NP}$. Sometimes the worst case instances and values cannot be computed either. Therefore in order to do the comparison, approximate values are used. For correctness, a lower bound of the optimal value and an upper bound of the worst case value are computed in such cases.

A very simple lower bound on the maximum completion time is the ideal machine load $\bar{H}$, which is given by the ratio of the total processing time and the number of machines $\bar{H} = \sum_{j=1}^{n} p_j/m$. Although this lower bound is not very tight, it can be used to preliminary evaluate the performance of our heuristic through *relative performance* ratio (10).

3 METHODOLOGY AND STRATEGY

The basic concept of the methodology developed is the combination of several dispatching rules, proposed earlier in [13] along with the Critical Path Method (CPM) [22], so that schedules are created with respect to the shortest *makespan* C_{max} of the production plan. The CPM guarantees the revelation of the optimal schedule, which coincides to the shortest feasible path. However this method presumes that the jobs are already loaded on the machines and are following a specific sequence. In addition, dispatching rules do not guarantee optimal schedules and cannot be applied alone on precedence constraint problems. Allocating a job j with duration time p_j at the time point $t = 0$ before job i does not imply necessarily that it can be connected at time point $t = p_j$. Dispatching rules just allocate the jobs one after another and thus they are not appropriate for precedence constraint problems.

Combining the CPM with dispatching rules for solving precedence constraint problems has the advantage of utilizing effective features of each approach in a single solution procedure. Moreover, the proposed solution procedure is "intuitively intelligent" since it examines the performance of several dispatching rule applications combined with a diagram analysis methodology. In particular, the solution approach is divided into two phases as follows:

Phase A: Loading the required jobs and developing a sequence between them through the application of heuristic nature dispatching rules.

Phase B: Timing the jobs in the machines through the application of CPM.

The usage of dispatching rules plays a two-fold role. Firstly, several different dispatching rules are used to allocate the jobs to machines and to create the sequence with respect to the precedence constraints. Secondly, the initial scheduling is "improved" by selecting the dispatching rule performed best in each case. As shown in the computational results section considering the same problem parameters and precedence constraints diagram, the dispatching rules performance is independent of the number of jobs, which is actually the size of the problem. In other words, if one apply the proposed solution procedure for restricted number of jobs and find the best performing dispatching rule, then the solution procedure considering the same precedence constraints guarantees to find high quality solutions for every problem of the same class, regardless of the size of the problem itself. There are cases in which this improvement can be equal to reduction of the *makespan* to less than the half of the initial *makespan*. However, along with the good performance, an other advantage of the proposed solution procedure, is the small computational labor required.

4 SOLUTION PROCEDURE

Given the model formulation and the strategy discussed earlier, we proceed to the development of a solution approach to tackle effectively and efficiently the production scheduling problem with complex precedence constraints in identical parallel machines environment. The proposed solution framework mainly consists of the following five steps:

Step 1: Read the data and create the diagram of precedence constraints for each product P_K.

Step 2: Load the jobs to the m parallel machines and create a sequence of job performances, with respect to precedence and other constraints. Note that this step is based on the application of several dispatching rules that are related either to the duration of jobs or to the precedence constraints. In particular, the following dispatching rules are applied and evaluated:

SIRO (Service in Random Order), allocates the jobs on the machines in random order.

LPT (Longest Processing Time first), gives priority to the job with the longest processing time.

SPT (Shortest Processing Time first), gives priority to the job with the shortest processing time.

CP (Critical Path), gives priority to the job with the highest level on the span tree, which is equivalent to the job with the highest queue of jobs waiting after it.

LNS (Largest Number of Successors), gives priority to the job with the greatest number of successors in the span tree overhaul.

LNSNL (Largest Number of Successors in Next Level), gives the priority to the greatest number of successors in the next level or equivalently to the greatest number of immediate successors. This rule is also known as **GNIS** or **LNIS** (greatest or largest number of immediate successors) [23].

A key feature of LNSNL rule is that seems to generate high quality solutions for out-tree precedence constraint diagrams, compared to others. It differs from the simple LNS because it does not examine all successors of every job, but only for those jobs belonging to the next level. The LNS rule gives priority to the job with the largest number of successors overhaul. However, the precedence constraints do not leave many choices regarding the sequence to be followed and in most cases the results do not differ from SIRO. On the contrary, LNSNL rule allows more options, because it only examines jobs of the next level.

Step 3: Apply the Forward Pass Method (FPM) and calculate the early start ES_i and finish times EF_i for all jobs $i \in N$ and the total *makespan* C_{max}. The job precedence is appropriately formulated into a graph network where nodes are represented by arcs and precedence constraints by direct links between arcs. Lastly, task duration is assigned to each node. The FPM mainly consist of the following steps:

- Iterate until all jobs $i \in N$ are labelled.
- The starting activity/job, which is an activity with no predecessors, is determined first.
- The early start time of the starting activity is set to zero.
- The arcs of the diagram, defining job precedence order, are followed forward and early start time of job i is set equal to early finish time of job j, were j is predecessor of job i $(j \to i)$.
- If job i has more than one predecessor then the early start time of next job i is equal to the maximum early finish time of all predecessors.

$$\begin{array}{l} ES_i = \max_{j:j\to i}\{EF_j\} = \max\{ES_j + p_j\} \\ EF_i = ES_i + p_i \end{array} \tag{11}$$

Step 4: Apply the Backward Pass Method (BPM) to calculate the late start LS_j, the finish time LF_j and the total slack time TS_j of all jobs $j \in N$. The BPM mainly consists of the following steps:

- Iterate until all jobs $j \in N$ are labelled.
- The finishing activity/job, which is an activity with no successors, is determined first.
- The latest finish time of finishing times is set to the earlier start time of the project, i.e. the calculated total *makespan* C_{max}. The opposite direction of the arcs on the network diagram is followed and the late finish time of job j is labelled equal to the late start of the next job k. Note that according to the precedence constraints j is predecessor for job k, $j \to k$.
- If j has more than one successors then the late finish time of previous job is equal to the minimum late finish time of all successors late start time:

$$\begin{array}{l} LF_j = \min_{k:j\to k}\{LS_k\} = \min\{LF_k - p_k\} \\ LS_j = LF_j - pj. \end{array} \tag{12}$$

Finally the total slack time TS_j of a job j is calculated as

$$TS_j = LS_j - ES_j = LF_j - EF_j. \tag{13}$$

Step 5: Create the schedule diagram. Repeat steps 2 to 4 using a different dispatching rule.

Step 6: Compare the total *makespan* C_{max} of each schedule, output the best schedule and terminate.

It is important to note that the most determinant factor for efficient deployment of the above solution procedure is the selection through the comparison among the most appropriate dispatching rule based on the nature and the characteristics of the network diagram of the problem considered. Although at step 6 the solution procedure is terminated, one can apply the best performed dispatching rule alone to a larger instance of the problem with the same characteristics and guarantee simultaneously to find high quality solutions, without the need to compare the performance of different dispatching rules. This is due to the fact that there is evident correlation between the dispatching rule and the nature (chain, complex out-tree or in-tree and so on) of the network diagrams. As proven in the subsequent section, the performance of the dispatching rules is independent to the problem size.

5 APPLICATIONS AND COMPUTATIONAL RESULTS

The proposed solution methodology is applied to various problem applications, which are instances of the above-described parallel machine production scheduling problem with different complexities of the precedence constraints. In the subsections below, the nature and the particular characteristics of each problem is presented along with the computational results obtained. Moreover, for each individual problem scaling issues are examined. In particular, the solution procedure is applied for different number of jobs n with the same precedence constraints and processing times, by changing the number of products G_k to be produced of each product type.

5.1 Problem (1)

The first problem involves three product types ($K = 3$) P_1, P_2 and P_3, in particular P_1 and P_3 of out-tree nature networks and P_2 of chain shape network diagram, as shown in Figure 1. Table 1 presents all related parameters of problem (1) and contains the number of jobs/production phases Φ_K, the precedence constraints between jobs and the processing times p_j for each job j of each product type K, respectively. The number of available identical parallel machines m is set to 10 and the production requirements for each product type K are $G_1 = 10$, $G_2 = 20$ and $G_3 = 15$, respectively. Therefore, the total number of products to be produced pp is equal to $\sum_{K=1}^{3} G_K = 45$ products and the total number of jobs n to be scheduled is equal to $\sum_{K=1}^{3} \Phi_K G_K = 190$ jobs.

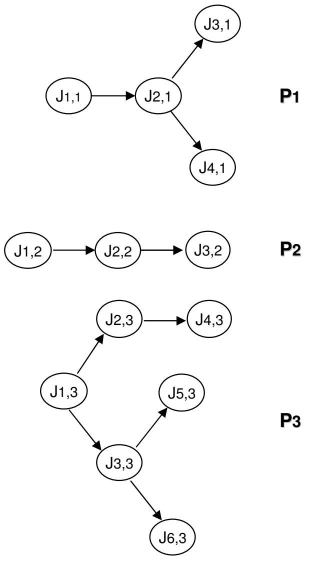


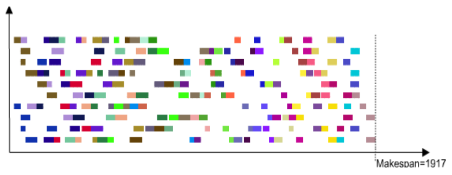
Fig. 1. Problem (1): Network diagrams for each product P_K

The schedules produced using the proposed solution procedure, given the data of Table 1, are illustrated in Figure 2, in the form of Gantt charts. In particular, Figures 2 a)–2 f) refer to the applications of SIRO, LPT, SPT, CP, LNS and LNSNL dispatching rules, respectively. The horizontal axis represents time and the vertical axis the machines m. Each bar shows the time schedule of each machine. Different

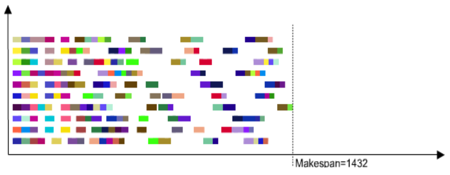
Product type/ Number of Phases	**Jobs**	**Precedence Constraints**		**Processing Time**
		Predecessors	Precedence jobs	
P_1 Φ_1=4	$J_{1,1}$	–	–	40
	$J_{2,1}$	1	$J_{1,1}$	35
	$J_{3,1}$	1	$J_{2,1}$	48
	$J_{4,1}$	1	$J_{2,1}$	45
P_2 Φ_2=3	$J_{1,2}$	–	–	45
	$J_{2,2}$	1	$J_{1,2}$	33
	$J_{3,2}$	1	$J_{2,2}$	25
P_3 Φ_3=6	$J_{1,3}$	–	–	34
	$J_{2,3}$	1	$J_{1,3}$	27
	$J_{3,3}$	1	$J_{1,3}$	52
	$J_{4,3}$	1	$J_{2,3}$	63
	$J_{5,3}$	1	$J_{3,3}$	49
	$J_{6,3}$	1	$J_{3,3}$	53

Table 1. Problem (1): Data and precedence constraints

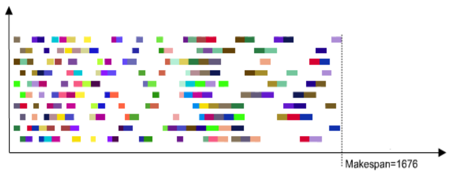
colors present processing of different jobs and white spaces between jobs processing show the time slack of each machine schedule. Lastly, the latest idle time of all machines denotes the total *makespan* $C_{max}^{heuristic}$ of the schedule (vertical dashed lines).



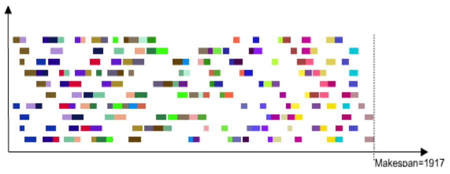
(a) SIRO (C_{max}^{SIRO}=1917)



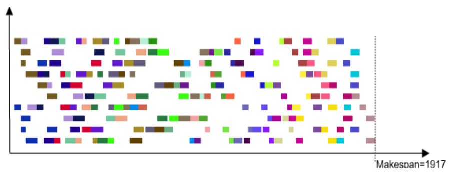
(b) LPT (C_{max}^{LPT}=1432)



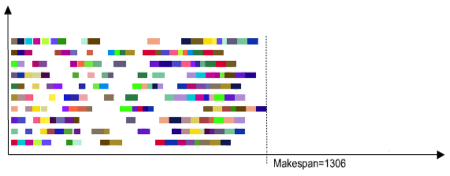
(c) SPT (C_{max}^{SPT}=1676)



(d) CP (C_{max}^{CP}=1917)



(e) LNS (C_{max}^{LNS}=1917)



(f) LNSNL (C_{max}^{LNSNL}=1306)

Fig. 2. Problem (1): Gantt charts of dispatching rules applications for the instance $G_1 = 10$, $G_2 = 20$ & $G_3 = 15$

Obviously, the application of LNSNL dispatching rule (see Figure 2 f)) produced the best schedule. The application of LNSNL minimized the total *makespan* C_{max}, the idle times of all machines and compressed all jobs to the left section of the time axis. On the contrary, Figures 2 a), 2 d) and 2 e) show the profiles of the schedule diagrams produced by applications of SIRO, CP and LNS dispatching rules along with the respective total *makespan* $C_{max}^{heuristic}$. These rules appear to have the worst performance, since idle times of all machines are very late (e.g. see the distances between the jobs executions on the charts) and overall quality of the schedules produced is very poor. On the other hand, Figure 2 b) shows that the application of the LPT rule significantly improves the quality of the resulted schedule, contrary to SIRO, CP and LNS dispatching rules.

Lastly, Figure 2 c) presents the schedule diagram and the total *makespan* produced by SPT, which seems slightly bigger than that of LPT rule. Recall that SPT, contrary to the Longest Processing Time first rule, considers the job with Shortest Processing Time first. It is worth mentioning that solutions generated from LNS have the same quality (i.e. schedule diagram and total *makespan*) with those produced by SIRO and CP rules. Note that this is one of the worst case performance of LNS and CP [24, 25] since CP and other precedence related dispatching rules have generated the best schedules on well-known problems, such as $Pm|p_j = 1; in - tree|C_{max}$ and $Pm|p_j = 1; out - tree|C_{max}$ [13].

Number of products produced G_K for each type K	$G_1 = 2$ $G_2 = 2$ $G_3 = 2$	$G_1 = 5$ $G_2 = 8$ $G_3 = 7$	$G_1 = 10$ $G_2 = 10$ $G_3 = 10$	$G_1 = 15$ $G_2 = 5$ $G_3 = 10$	$G_1 = 10$ $G_2 = 20$ $G_3 = 15$	$G_1 = 20$ $G_2 = 40$ $G_3 = 30$
Total Number of Jobs	$n = 26$	$n = 86$	$n = 130$	$n = 135$	$n = 190$	$n = 360$
Dispatching Rule	***Makespan*** ($C_{max}^{heuristic}$ in time units)					
SIRO	334	880	1 408	1 390	1 917	4 100
LPT	312	654	1 000	1 046	1 432	2 738
SPT	329	793	1 039	1 077	1 676	3 481
CP	334	880	1 408	1 390	1 917	4 100
LNS	334	880	1 408	1 390	1 917	4 100
LNSNL	**270**	**586**	**899**	**914**	**1 306**	**2 552**
Ideal machine load ($\bar{H}$)	109.8	361	549	581.5	791	1 582
CPU time (sec)	**0.3**	$\approx$ **1.5**	$\approx$ **3**	$\approx$ **3**	$\approx$ **6**	$\approx$ **21**
Dispatching Rule	***Relative Performance*** ($C_{max}^{heuristic}/C_{max}^{H}$)					
SIRO	3.04	2.44	2.56	2.39	2.42	2.59
LPT	2.84	1.81	1.82	1.80	1.81	1.73
SPT	3.00	2.20	1.89	1.85	2.12	2.20
CP	3.04	2.44	2.56	2.39	2.42	2.59
LNS	3.04	2.44	2.56	2.39	2.42	2.59
LNSNL	**2.46**	**1.62**	**1.64**	**1.57**	**1.65**	**1.61**

Table 2. Problem (1): Results and performance of dispatching rules for various total number of jobs n and constant $m = 10$

Table 2 summarizes on the fifth column the results (total *makespan* $C_{max}^{heuristic}$ in unit times) obtained by application of the proposed solution procedure to the given production requirements. However, in order to examine scaling issues, the proposed solution procedure is applied to several instances with the same precedence constraints and processing times (as those shown in Table 1), although for different number of jobs n, by changing the number of products G_K required for production from each product type K. Evidently, using the same problem parameters set and precedence constraints diagrams, the dispatching rules performance is independent to the total number of jobs n, which is actually the size of the problem.

Therefore, one can apply the proposed solution procedure to an instance with restricted total number of jobs n in order to determine quickly the best performing dispatching rule, and later on apply solely the best performing rule along with the proposed solution procedure to a larger instance. Due to the fact that when considering the same precedence constraints and processing times, the best performing dispatching rule is expected to provide high quality solutions, regardless of the size of the problem itself.

Furthermore, Table 2 contains the *relative performance* of each dispatching rule using equation (10) and as lower bound the ideal machine load $\bar{H}$. It is worth mentioning that as the total number of jobs n increases, the difference between the best schedule and worst performing dispatching rules is higher. On the contrary, the relative performance of the best performing dispatching rule seems to be constant as the size of the problem increases.

5.2 Problem (2)

The second problem involves two product types ($K = 2$) P_1 and P_2, both of out-tree nature network diagrams, as shown in Figure 3. Although products P_1 and P_3 of problem (1) assumed also out-tree nature network diagrams, those of the second problem are more complex. Table 3 demonstrates all related parameters of problem (2), including the number of jobs/production phases Φ_K, the precedence constraints among jobs and processing time p_j for each job j of both product types. The number of items to be produced from each product type K are $G_1 = 10$ and $G_2 = 15$ and the number of available identical parallel machines m is 10. Thus, the total production requirements pp is 25 and the total number of jobs n to be scheduled is 245.

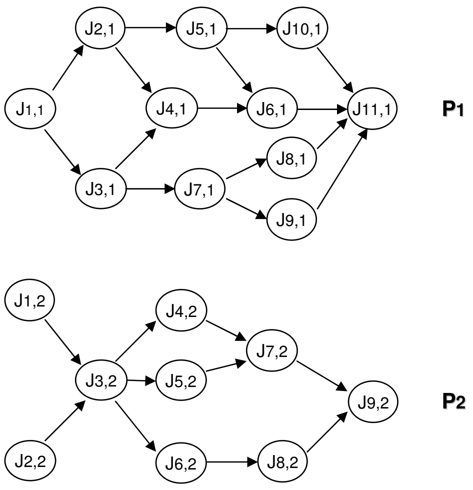


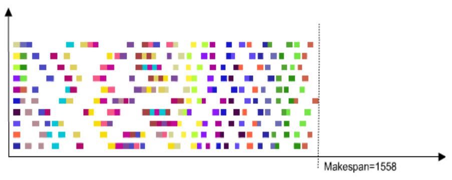
Fig. 3. Problem (2): Network diagrams for each product P_K

Product type/ Number of Phases	**Jobs**	**Precedence Constraints**		**Processing Time**
		Predecessors	Precedence jobs	
P_1 Φ_1=11	$J_{1,1}$	–	–	27
	$J_{2,1}$	1	$J_{1,1}$	35
	$J_{3,1}$	1	$J_{1,1}$	32
	$J_{4,1}$	2	$J_{2,1}$ & $J_{3,1}$	25
	$J_{5,1}$	1	$J_{2,1}$	37
	$J_{6,1}$	2	$J_{4,1}$ & $J_{5,1}$	23
	$J_{7,1}$	1	$J_{3,1}$	34
	$J_{8,1}$	1	$J_{7,1}$	25
	$J_{9,1}$	1	$J_{7,1}$	27
	$J_{10,1}$	1	$J_{5,1}$	38
	$J_{11,1}$	4	$J_{6,1}$, $J_{8,1}$, $J_{9,1}$ & $J_{10,1}$	32
P_2 Φ_2=9	$J_{1,2}$	–	–	35
	$J_{2,2}$	–	–	20
	$J_{3,2}$	2	$J_{1,2}$ & $J_{2,2}$	22
	$J_{4,2}$	1	$J_{3,2}$	30
	$J_{5,2}$	1	$J_{3,2}$	25
	$J_{6,2}$	1	$J_{3,2}$	27
	$J_{7,2}$	2	$J_{4,2}$ & $J_{5,2}$	30
	$J_{8,2}$	1	$J_{6,2}$	25
	$J_{9,2}$	2	$J_{7,2}$ & $J_{8,2}$	32

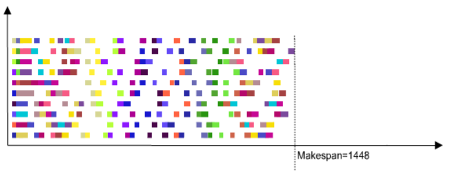
Table 3. Problem (2): Data and precedence constraints

The schedules produced using the proposed solution procedure given the above production requirements are illustrated in the form of Gantt charts, as shown in Figure 4. In particular, Figures 4 a) and 4 b) demonstrate the schedules produced by applications of LPT and SPT dispatching rules, respectively. Contrary to Problem (1), SPT produces a slightly improved schedule compared to LPT dispatching rule in terms of total *makespan*. Moreover, Figure 4 c) shows the Gantt chart schedule produced by applications of CP, LNS and SIRO dispatching rules. Clearly, the schedule produced is insufficient in terms of quality since the idle time of machines is high due to the large spaces (time slack) occurred between job's execution. Lastly, Figure 4 d) demonstrates the schedule produced by application of LNSNL dispatching rule. Obviously, LNSNL is again the best performing dispatching rule. The application of LNSNL minimized the total *makespan* C_{max}, the idle times of all machines and compressed all jobs to the left section of the time axis.

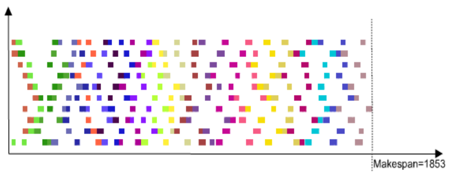
Table 4 summarizes on the fourth column the results (total *makespan* $C_{max}^{heuristic}$ in unit times) obtained by the proposed solution procedure for the instance $G_1 = 10$, $G_2 = 15$ and $m = 10$. Similarly, scaling issues are examined. The rest of columns contain detailed results for each dispatching rule, for several instances (different total number of jobs n and number of products produced pp from each product type K). It is clear that the performance of dispatching rules is independent to the size of the



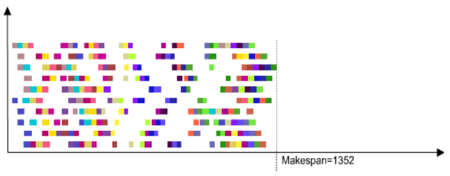
(a) LPT (C_{max}^{LPT}=1558)



(b) SPT (C_{max}^{SPT}=1448)



(c) CP, LNS & SIRO ($C_{max}^{CP,LNS,SIRO}$ = 1 853)



(d) LNSNL ($C_{max}^{LNSNL} = 1\,352$)

Fig. 4. Problem (2): Gantt charts of dispatching rules applications for the instance $G_1 = 10\,\&\,G_2 = 15$

Number of products produced G_K	$G_1 = 2$ $G_2 = 2$	$G_1 = 5$ $G_2 = 5$	$G_1 = 10$ $G_2 = 10$	$G_1 = 10$ $G_2 = 15$	$G_1 = 20$ $G_2 = 30$
Total Number of Jobs	$n = 40$	$n = 100$	$n = 200$	$n = 245$	$n = 490$
Dispatching Rule	***Makespan*** (C_{max} in time units)				
SIRO	390	862	1 541	1 853	3 611
LPT	306	707	1 249	1 558	2 935
SPT	307	623	1 198	1 448	2 885
CP	390	862	1 541	1 853	3 611
LNS	390	862	1 541	1 853	3 611
LNSNL	**263**	**607**	**1 078**	**1 352**	**2 653**
Ideal machine load ($\bar{H}$)	116.2	290.5	581	704	1 408
CPU time (sec)	$\approx$ **0.5**	$\approx$ **2**	$\approx$ **7**	$\approx$ **10**	$\approx$ **37**
Dispatching Rule	***Relative Performance*** ($C_{max}^{heuristic}/C_{max}^{H}$)				
SIRO	3.36	2.97	2.65	2.63	2.56
LPT	2.63	2.43	2.15	2.21	2.08
SPT	2.64	2.14	2.06	2.06	2.05
CP	3.36	2.97	2.65	2.63	2.56
LNS	3.36	2.97	2.65	2.63	2.56
LNSNL	**2.26**	**2.09**	**1.86**	**1.92**	**1.88**

Table 4. Problem (2): Results and performance of dispatching rules for various total number of jobs n and constant $m = 10$

problem (total number of jobs n). Moreover, the performance of dispatching rules in both problems examined so far, regardless the different degree of complexity, is almost constant. The best schedule corresponds to the LNSNL rule, while SIRO, CP and LNS dispatching rules perform worst producing schedules with the largest total *makespan*.

As already mentioned, problems (1) and (2) differ only in the degree of complexity of their precedence constraints network diagrams. However, both belong to the same class of problems with out-tree precedence network diagram constraints. Therefore, the precedence related dispatching rules are expected to generate the best schedules. On the contrary, as will be discussed later, applying rules related to the precedence constraints is meaningless for chain network diagrams.

5.3 Problem (3)

So far our solution procedure has been applied only on simple and complex out-tree precedence constraints network diagrams. However, apart from scaling issues, there is a need to examine the differences between other classes of network diagrams. Therefore, we assume that the third problem's network diagrams of products are simple chains. In particular, two product types ($K = 2$) (P_1 and P_2) of chain nature network diagrams are involved as those shown in Figure 5. Table 5 contains all related data and parameters concerning problem (3). The production requirements are initially set to G_1=15 and $G_2 = 12$. The number of the available identical machines m is 10; therefore, the total number of products to be produced is $pp = 27$ and the total number of jobs to be scheduled is $n = 201$.

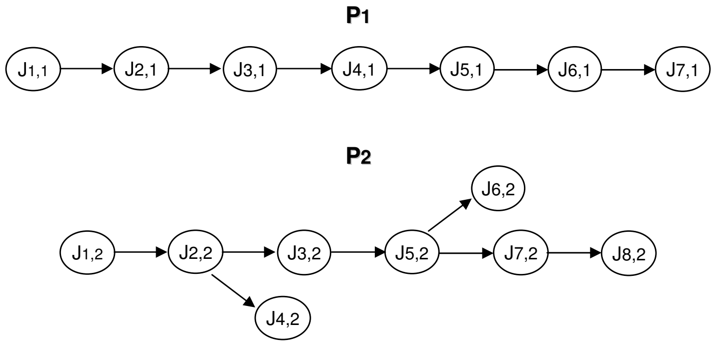


Fig. 5. Problem (3) network diagrams for each product P_K

The results obtained for each dispatching rule application given the above production requirements are illustrated in the form of Gantt charts, as shown in Figure 6. Figure 6 b) presents the schedules generated from SIRO, LNS and CP dispatching rules. Still the schedule produced from the random order of jobs con-

Product type/ Number of Phases	**Jobs**	**Precedence Constraints**		**Processing Time**
		Predecessors	Precedence jobs	
P_1 Φ_1=7	$J_{1,1}$	–	–	23
	$J_{2,1}$	1	$J_{1,1}$	16
	$J_{3,1}$	1	$J_{2,1}$	12
	$J_{4,1}$	1	$J_{3,1}$	25
	$J_{5,1}$	1	$J_{4,1}$	32
	$J_{6,1}$	1	$J_{5,1}$	12
	$J_{7,1}$	1	$J_{6,1}$	32
P_2 Φ_2=8	$J_{1,2}$	–	–	25
	$J_{2,2}$	–	$J_{1,2}$	32
	$J_{3,2}$	1	$J_{2,2}$	20
	$J_{4,2}$	1	$J_{2,2}$	19
	$J_{5,2}$	1	$J_{3,2}$	17
	$J_{6,2}$	1	$J_{5,2}$	21
	$J_{7,2}$	1	$J_{5,2}$	13
	$J_{8,2}$	1	$J_{7,2}$	23

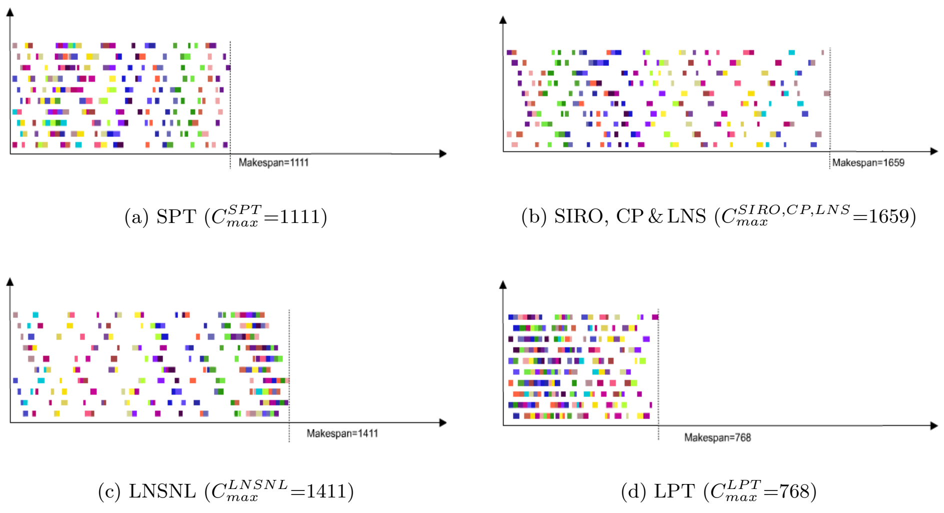
Table 5. Problem (3) data and precedence constraints

cedes to be the worst one. Furthermore, Figures 6 a) and 6 c) illustrate the schedules produced from SPT and LNSNL dispatching rules, respectively. Contrary to problems (1) and (2), LNSNL rule does not lead to the best schedule when applied to problem (3). On the other hand, LPT dispatching rule produced the best schedule (see Figure 6 d)). Moreover, the total *makespan* C_{max} produced from LPT is reduced to less than the half, compared to other dispatching rules schedules, such as that of SIRO, LNS and CP.

Therefore, as expected, the performance of dispatching rules is sensitive to the precedence constraints network diagrams. The rules performed well on complicated out-tree network diagrams, it is not necessary to have the same performance for chain network diagrams. In addition to the problems examined previously, the performance of dispatching rules remains the same for various total number of jobs n and production requirements G_k (see Table 6). Furthermore, the *relative performance* of each dispatching rule except very small instances is almost constant. This leads to the conclusion that the performance of dispatching rules, and therefore the proposed solution procedure, is independent of problem size. It is worth mentioning that execution times shown in Tables 2, 4 and 6 were obtained when all dispatching rules are being applied sequentially. Therefore, execution time is expected to be significantly reduced, if the best performing rule is applied solely.

6 CONCLUSIONS

In this paper we presented a procedure for solving an $\mathcal{NP}$-hard real-life production scheduling problem with complex precedence constraints in an identical parallel ma-



(a) SPT (C_{max}^{SPT}=1111) (b) SIRO, CP & LNS ($C_{max}^{SIRO,CP,LNS}$=1659)

(c) LNSNL (C_{max}^{LNSNL}=1411) (d) LPT (C_{max}^{LPT}=768)

Fig. 6. Problem (3): Gantt charts of dispatching rules applications for the instance $G_1 = 15 \,\&\, G_2 = 12$

Number of Products produced G_K	$G_1 = 5$ $G_2 = 5$	$G_1 = 15$ $G_2 = 12$	$G_1 = 30$ $G_2 = 24$	$G_1 = 50$ $G_2 = 50$
Total Number of Jobs	$n = 75$	$n = 201$	$n = 402$	$n = 750$
Dispatching Rule	***Makespan*** (C_{max} in time units)			
SIRO	667	1 659	3 234	5 821
LPT	**318**	**768**	**1 487**	**2 746**
SPT	447	1 111	2 204	4 194
CP	667	1 659	3 234	5 821
LNS	667	1 659	3 234	5 821
LNSNL	510	1 411	2 522	4 048
Ideal machine load ($\bar{H}$)	161	432	864	1 610
CPU time (sec)	**≈ 1.3**	**≈ 8**	**≈ 35.7**	**≈ 120.7**
Dispatching Rule	***Relative Performance*** ($C_{max}^{heuristic}/C_{max}^{H}$)			
SIRO	4.14	3.84	3.74	3.62
LPT	**1.98**	**1.78**	**1.72**	**1.71**
SPT	2.78	2.57	2.55	2.60
CP	4.14	3.84	3.74	3.62
LNS	4.14	3.84	3.74	3.62
LNSNL	3.17	3.27	2.92	2.51

Table 6. Problem (3): Results and performance of dispatching rules for various total number of jobs n and constant $m = 10$

chines environment, in adequately short computational time. The basic concept of our solution procedure is based on comparison of the performance of several dispatching rules, combined with the Critical Path Method. As discussed earlier, this solution approach can solve problems consisting of simple chains as well as complex in-trees and out-trees precedence constraint diagrams. The results presented indicated that the performance of the solution procedure is very efficient in terms of computational effort and, most importantly, independent of the size of the problem (products and jobs) and the complexity of the precedence constraints.

Furthermore, our results provided valuable information regarding the performance of several dispatching rules on different precedence networks. In particular, for complex precedence constraints problems, precedence related dispatching rules, such as the LNSNL, generated the best schedules, while for chain precedence networks processing time related rules seem to work better. In fact, there were cases where the makespan was reduced to more than the half when different rules were applied. It is quite satisfactory in all cases studied in this paper that the computational time required varied from less than a second to only a few seconds, depending on the problem size and precedence constraints, even for up to 360 jobs. This fact is a prerequisite for deployment of production scheduling solutions methods in actual industrial environments, where users require fast solutions, to be able to adjust production plans and schedules, if necessary. The real world is dynamic, thus effective and efficient decision support tools are needed to help practitioners run their daily operations in the best possible manner.

Acknowledgements

The author is indebted to the anonymous reviewers for their useful comments and suggestions that helped improve the work presented in this paper.

REFERENCES

[1] Hou, E. S.—Ansari, N.—Ren, H.: A Genetic Algorithm for Multiprocessor Scheduling. IEEE Transactions on Parallel and Distribution Systems, Vol. 2, 1994, No. 5, pp. 113–120.

[2] Garey, M. R.—Johnson, D. S.: Computers and Intractability. N.H. Freeman, New York, 1979.

[3] Karp, R. M.: Reducibility Among Combinatorial Problems. In: R.E. Miller and J.W. Thatcher (Eds.): Complexity of Computations, Plenum Press, New York, 1972.

[4] Bilge, Ü.—Klraç, F.—Kurtulan, M.—Pekgün, P.: A Tabu Search Algorithm for Parallel Machine Total Tardiness Problem. Computers & Operations Research, Vol. 31, 2004, No. 3, pp. 397–414.

[5] Timkovsky, V. G.: Identical Parallel Machines vs. Unit-Time Shops and Preemptions vs. Chains in Scheduling Complexity. European Journal of Operational Research, Vol. 149, 2003, No. 2, pp. 355–376.

[6] Lepere, R.—Mounie, G.—Trystram, D.: An Approximation Algorithm for Scheduling Trees of Malleable Tasks. European Journal of Operational Research, Vol. 142, 2002, No. 2, pp. 242–249.

[7] Hurink, J.—Knust, S.: List Scheduling in a Parallel Machine Environment with Precedence Constraints and Setup Times. Operations Research Letters, Vol. 29, 2001, No. 5, pp. 231–239.

[8] Bampis, E.—Giroudeau, R.—Konig, J. C.: An Approximation Algorithm for the Precedence Constrained Scheduling Problem with Hierarchical Communications. Theoretical Computer Science, Vol. 290, 2003, No. 3, pp. 1883–1895.

[9] Belouadah, H.—Potts, C. N.: Scheduling Identical Parallel Machines to Minimize Total Weighted Completion Time. Discrete Applied Mathematics, Vol. 48, 1995, pp. 201–218.

[10] Murovec, B.—Suhel, P.: A Repairing Technique for the Local Search of the Job-Shop Problem. European Journal of Operational Research, Vol. 153, 2004, No. 1, pp. 220–238.

[11] Schuster, C. J.—Framinan, J. M.: Approximative Procedures for No-Wait Job Shop Scheduling. Operations Research Letters, Vol. 31, 2003, No. 4, pp. 308–318.

[12] Tarantilis, C. D.—Kiranoudis, C. T.: A Modern Local Search Method for Operations Scheduling of Dehydration Plants. Journal of Food Engineering, Vol. 52, 2002, No. 1, pp. 17–23.

[13] Pinedo, M.: Scheduling Theory, Algorithms and Systems. Prentice Hall, Englewood Cliffs, NJ, 1995.

[14] Dunstall, S.—Wirth, A.: Heuristic Methods for the Identical Parallel Machine Flowtime Problem with Set-Up Times. Computers & Operations Research, Vol. 32, 2005, pp. 2479–2491.

[15] Gendreau, M.—Laporte, G.—Guimaraes, E. M.: A Divide and Merge Heuristic for the Multiprocessor Scheduling Problem with Sequence Dependent Setup Times. European Journal of Operational Research, Vol. 133, 2001, pp. 183–189.

[16] França, P.—Gendreau, M.—Laporte, G.—Müller, F.: A Tabu Search Heuristic for the Multiprocessor Scheduling Problem with Sequence Dependent Setup Times. International Journal of Production Economics, Vol. 43, 1996, pp. 79–89.

[17] Cowling, P. I.—Kendall, G.—Soubeiga, E.: Relating Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. In: Cangnoni et al. (Eds.): Applications of Evolutionary Computing: Evo Workshops, LNCS 2279, Springer, 2002, pp. 1–10.

[18] Pinedo, M.—Chao, X.: Operations Scheduling and Applications in Manufacturing and Services. Irwin/McGraw Hill, Boston, 1999.

[19] Tarantilis, C. D.—Kiranoudis, C. T.: A List-Based Threshold Accepting Method for the Job Shop Scheduling Problem. International Journal of Production Economics, Vol. 77, 2002, No. 2, pp. 159–171.

[20] Tarantilis, C. D.—Kiranoudis, C. T.—Vassiliadis, V. S.: A List Based Threshold Accepting Metaheuristic for the Heterogeneous Fixed Fleet Vehicle Routing Problem. Journal of the Operational Research Society, Vol. 54, 2003, No. 1, pp. 65–71.

[21] Tarantilis, C. D.—Kiranoudis, C. T.—Vassiliadis, V. S.: A Threshold Accepting Meta-Heuristic for the Heterogeneous Fixed Fleet Vehicle Routing Problem. European Journal of Operational Research, Vol. 152, 2004, No. 1, pp. 148–158.

[22] De Reyck, B.—Herroelen, W.: A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Generalized Precedence Relations. European Journal of Operational Research, Vol. 111, 1998, No. 1, pp. 152–174.

[23] Yang, K. K.: A Comparison of Dispatching Rules for Executing a Recourse-Constrained Project with Estimated Activity Durations. Omega, Vol. 26, 1998, No. 6, pp. 729–738.

[24] Singh, G.—Zinder, Y.: Worst-Case Performance of Critical Path Type Algorithms. International Transactions in Operational Research, Vol. 7, 2000, No. 5, pp. 383–399.

[25] Singh, G.: Performance of Critical Path Type Algorithms for Scheduling on Parallel Processors. Operations Research Letters, Vol. 29, 2001, No. 1, pp. 17–30.

Katerina El Raheb holds a diploma in chemical engineering from the National Technical University of Athens and works on behalf of a systems engineering company as manufacturing execution systems consultant. She has participated in various projects that involve implementation of production management systems, design and development of custom software applications, batch tracking, data acquisition, production engineer training, integration within ERP systems and supply chain management systems.

Panagiotis P. Repoussis is a research assistant at the Department of Management Science & Technology of Athens University of Economics & Business. He received his diploma in chemical engineering from National Technical University of Athens and holds an MSc in process systems engineering from Imperial College London. His research interests apply in the fields of transportation and logistics, decision support systems, combinatorial optimization and approximation algorithms.

Christos D. Tarantilis is a lecturer at Athens University of Economics & Business. He received his MSc and Ph. D. in operations research from London School of Economics and National Technical University of Athens, respectively. His research concentrates on quantitative methods of operations management, logistics and transportation systems. He has more than 60 publications in leading academic journals, books and conferences and he is recipient of Best Teaching Faculty Awards. Furthermore, he is member of Board of Directors and technical consultant for public organizations and private industries.

Chris T. Kiranoudis is an assistant professor of process design and systems analysis at the School of Chemical Engineering, National Technical University of Athens. He holds a diploma degree in chemical engineering and a Ph. D. degree in non-linear mathematical programming from the same university. He has more than 100 publications in leading leading academic journals. His current research focuses on design and control of chemical processes and plants, mathematical programming, real-time and dynamic optimization, and development of information systems.