Partially-Concurrent Open Shop Scheduling

Tal Grinshpoun · Hagai Ilani · Elad Shufan

Abstract The *partially-concurrent* open shop scheduling problem is presented. The standard open shop scheduling problem is generalized by allowing some operations to be processed concurrently. This generalization is directly motivated from a real-life timetabling project of assigning technicians to airplanes in an airplane garage. A schedule for the partially-concurrent problem is represented by a digraph. We show that the scheduling problem is equivalent to a problem of orienting a given undirected graph, called a conflict graph. The schedule digraph is then modeled by a matrix, generalizing the rank matrix representation. The problem is shown to be NP-Hard. The rank matrix representation is also used in an algorithm that heuristically constructs an open shop schedule.

Keywords Open Shop Scheduling · Concurrent machines · Technician timetabling· Graph orientation

1 Introduction

An open shop scheduling (OSS) problem consists of n jobs that should be processed on m machines. An operation (i,j) refers to the processing of job i = 1, 2,.. . , n in machine j = 1, 2,... , m. The processing time of operation

Tal Grinshpoun
Department of Industrial Engineering and Management, Ariel University, Ariel, Israel
E-mail: talgr@ariel.ac.il

Hagai Ilani
Department of Industrial Engineering and Management, SCE – Shamoon College of Engineering, Ashdod, Israel
E-mail: hagai@sce.ac.il

Elad Shufan
Physics Department, SCE – **Shamoon College of Engineering, Beer-Sheva, Israel E-mail: elads@sce.ac.il**

(i, j) is denoted by p_{ij}. In a standard OSS every job visits one machine at a time, and every machine hosts only one job at a time.

Shop scheduling problems were originally designed for machines. Nevertheless, there are many task scheduling applications in which employees are represented as the machines; hence the resulting solution is a set of timetables (schedules) for the employees.

An important aspect of OSS concerns the mathematical representation of a given schedule. Bräsel and Kleinau have introduced the rank matrix representation [7,5], which is significant for both theory and practice. A major advantage of the rank matrix representation when compared to its alternative [16] is the one-to-one correspondence between a matrix and a schedule, provided that the schedule is semi-active [18], for which operations are performed as early as possible once the order of processing is known. Constructive heuristic algorithms were suggested, based on rank matrices [10]. Rank matrix procedures were applied to neighbourhood definitions in local search algorithms, including those of crossover and mutations in genetic algorithms [1]. Among the theoretical achievements that are based on the idea of rank matrix are the irreducibility theory [2,8] as well as complexity analysis of some special OSS problems [9].

In this work we extend the rank matrix representation to a problem of *partially-concurrent* open shop scheduling (PCOSS). In a PCOSS problem some operations are allowed to be processed concurrently, while some are not. In the PCOSS presented herein preemption is not allowed, and no due or release dates are given. A variety of previously discussed issues can be extended by the more general PCOSS matrix representation. Several such extensions are discussed in this article.

A related problem previously presented in the literature is that of *concurrent* open shop scheduling [21,17,15]. It is also referred to as a parallel machine environment with m fully dedicated machines [14], denoted *PDm*. In a *PDm*, a job can be split to be simultaneously processed on several machines. The two extremes of PCOSS are the standard OSS, where operations of the same job are never processed concurrently, and the fully-concurrent open problem, where all the operations of a given job are allowed to be processed concurrently. The *PDm* notation does not reflect the connection between the concurrent open shop and the standard one. We suggest that for all the discussed types of an OSS, the machine environment field α of Graham's $\alpha|\beta|\gamma$ classification scheme [12] will include *O* (or *Om* when the number of machines should be specified explicitly). Concurrency issues will be included in the β field, with the "*conc*" entry for the concurrent open shop case, and "*pconc*" when a partially concurrent open shop is considered.

Roemer discusses the complexity of $O|conc|\sum w_i C_i$ [19], where C_i is the completion time of job i, and w_i the corresponding weight. It is shown that the general problem is NP-hard in the strong sense. Minimizing the makespan $C_{max} = max\{C_i \mid 1 \leq i \leq n\}$, which is commonly considered in a standard OSS, is not considered in the concurrent version. This is due to its over simplicity: any semi-active schedule will be optimal with $C_{max} =$

$max\{\sum_{k=1}^{n} p_{kj} \mid 1 < j < m\}$. For the general PCOSS problem the complexity of a schedule designed to minimize the makespan should be discussed, with $O|conc|C_{max}$ being trivially polynomial, and a general $O||C_{max}$ being NP-Hard.

The PCOSS problem is more general than the standard OSS and can therefore describe a large variety of real-life scenarios. In fact, the present study is directly motivated from a timetabling project of assigning technicians to airplanes in an airplane garage. The airplane garage task scheduling problem is mentioned in the literature with respect to both the standard [8] and the concurrent [21] OSS versions. Consider a fleet of airplanes. A set of tasks should be performed on each plane in order to prepare it for action. Every task is done by a technician who has the expertise to do this task alone. In reality, some tasks can be performed simultaneously on a plane, e.g., while one technician checks the engine, another technician can check the wing. But other tasks exist that disturb each other, and therefore cannot be performed concurrently, which is similar to the standard open shop. Indeed, in the timetabling project that inspired the present research, some of the technicians' tasks could be performed in parallel. The $O|pconc|f_{obj}$, with any objective function f_{obj}, naturally describes this scenario, with airplanes corresponding to jobs, and tasks (or technicians) corresponding to machines.

In section 2 the generalization of the rank matrix representation is given. Section 3 deals with the complexity of PCOSS. A constructive algorithm, first suggested with respect to the standard OSS [10], is extended to the PCOSS problem in section 4, followed by experiments (section 5). A discussion is given in section 6.

2 Matrix representation of Partially-Concurrent Open Shop Scheduling

The standard OSS is reviewed first. A scheduling problem consists of a set of n jobs J_i, where $i \in I = \{1, 2, \ldots, n\}$, which should be processed on a set of m machines M_j, $j \in J = \{1,2,\ldots,m\}$. In addition, the processing times of the operations are listed as elements of an $n \times m$ matrix $PT = [p_{ij}]$, with p_{ij} denoting the processing time of an operation (i, j). In a general scenario the jobs might visit only a partial set of machines. For clarity of presentation we assume that every job visits all the machines, i.e., $p_{ij} > 0$ for every $i \in I$ and $j \in J$. In the discussion (Section 6) we explain how to treat the general scenario. In a non-concurrent OSS two operations of the same job cannot be processed concurrently. Therefore, a given schedule necessarily defines an order between operations of the same job (machine order). Similarly, a schedule sets an order between operations of the same machine (job order). Operations that do not share either a job or a machine can obviously be processed concurrently. Therefore, the schedule does not imply any order between such operations. This is illustrated in the following OSS example.

Example 1 In this example we consider $n = 3$ jobs and $m = 4$ machines, and the following processing times matrix:

$$PT = \begin{pmatrix} 21 & 40 & 6 \\ 30 & 18 & 35 \\ 15 & 8 & 25 \\ 28 & 8 & 24 \end{pmatrix} \quad (1)$$

A possible schedule for this instance is given in Figure 1(a) by its Gantt chart. Jobs J_1, J_2, and J_3 are recognized by their colours: dark red, white, and light green, respectively. In the given schedule the machine order of Job 1, taken as an example, is $M_4 \rightarrow M_2 \rightarrow M_1 \rightarrow M_3$. For machine 1, the job order is $J_2 \rightarrow J_1 \rightarrow J_3$. Both orders are easily read from the Gantt chart for this compact example.

The machine and job orders can be represented by a non-cyclic digraph called a *sequence graph* [7,6] – every operation is represented by a vertex (i, j) and each pair of consecutive operations (vertices) are connected by an arrow. The sequence graph that corresponds to the given schedule of example 1 is shown in Figure 1(b).

(a) (b)

Fig. 1 A possible schedule for an OSS with 3 jobs and 4 machines is given by (a) a machine-oriented Gantt chart, and (b) a sequence graph.

A convenient matrix representation for sequence graphs was suggested [7, 5]. The matrix, termed a rank matrix, is denoted by R. An entry r_{ij} is equal to the number of vertices in the longest path from a source to the vertex (i, j). A source is a vertex with zero indegree. It represents an operation that is scheduled first. The rank matrix of the schedule given in Figure 1 is

$$R = \begin{pmatrix} 2 & 4 & 1 \\ 1 & 3 & 2 \\ 5 & 3 & 4 \\ 4 & 2 & 3 \end{pmatrix} \quad (2)$$

The machine order or job order are both represented by the order of the entries of the relevant row or column, respectively. If $r_{ij} > 1$, then in row i or column j there exists an entry that equals $(r_{ij} - 1)$. Operation (i,j) is scheduled after the operation that corresponds to this entry. According to the "Latin-rectangle theorem" [7], there is a one-to-one correspondence between

a unique Latin rectangle and a semi-active schedule: Consider a matrix of size $n \times m$ whose elements are taken from the set $\{1, 2, \ldots, q\}$. If in each row and each column no entry appears more than once, the matrix is called a Latin rectangle, denoted $LR(n, m, q) = [lrij]$. The Latin rectangle $LR(n, m, q)$ corresponds to a schedule of an open shop with n jobs and m machines, if for every entry $lrij = 6\ 1$ there exists an entry $(lrij - 1)$ in either row i or column j.

In this article we show that the above ideas can be naturally extended to the case of PCOSS. An instance of a PCOSS contains, in addition to PT, a concurrence graph, or its complement, a conflict graph. These graphs describe whether pairs of operations may be processed concurrently or not. More precisely, the concurrence graph, CG, has $I \times J$ as its vertex set, and two operations (i, j) and (k, l) are adjacent if they may be processed concurrently. In the conflict graph, CG, (i, j) and (k, l) are adjacent if they may not be processed concurrently.

To summarize, for any pair of operations:

– if they relate to different machines and different jobs the pair is an edge of the concurrence graph.
– if they relate to the same machine and different jobs the pair is an edge of the conflict graph.
– if they relate to different machines and to the same job the pair can be either an edge of the concurrence graph or the conflict graph, depending on whether the pair can be processed concurrently or not (respectively). A pair of operations that is an edge of the conflict graph will be called a *conflict pair*. Given an operation (i,j), we say that operation (k, l) is a *conflicting operation* if the pair $\{(i, j), (k, l)\}$ is a conflict pair.

In Figure 2 we show an example of CG and CG for 3 jobs and 4 machines. For clarity, arrows connecting two operations that do not share a machine or a job are not shown in CG.

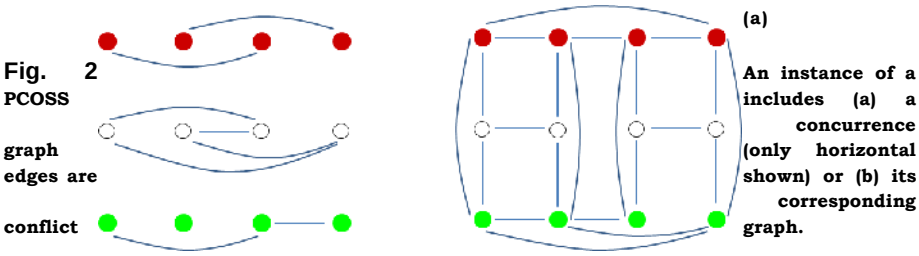

Fig. 2 An instance of a PCOSS includes (a) a graph concurrence edges are (only horizontal shown) or (b) its conflict corresponding graph.

A schedule for the PCOSS is given by the $n \times m$ matrix $ST = [stij]$, of the starting times of all the operations, or by the matrix $CT = [ctij]$, of the

completion times of all the operations. Obviously, $CT = ST + PT$. A schedule is *feasible* if for any conflict pair, $\{(i,j), (k,l)\}$, either $ct_{ij} \leq st_{kl}$ or $ct_{kl} \leq st_{ij}$. A given feasible schedule naturally defines an orientation of the conflict graph: an edge $\{(i,j), (k,l)\}$ will be oriented, $(i,j) \rightarrow (k,l)$, if $ct_{ij} \leq st_{kl}$ and $(k, l) \rightarrow (i,j)$, if $ct_{kl} \leq st_{ij}$. The resulting digraph, DC, is acyclic, because a cycle $(i,j) \rightarrow (k,l) \rightarrow \ldots \rightarrow (i,j)$ in DC means $ct_{ij} \leq st_{kl} < ct_{kl} \leq \ldots \leq st_{ij}$, which is impossible – an operation cannot be completed before it starts.

An acyclic orientation of the conflict graph will be called a *partial-sequence graph*. It generalizes the sequence graph representation.

Lemma 1 *A partial-sequence graph defines a schedule with the property that* $\{(i,j), (k,l)\}$ *implies* $ct_{ij} \leq st_{kl}$. *The schedule defined is unique, assuming semi-activeness.*

Proof Given a partial-sequence graph, DC, a schedule is defined inductively step by step as follows: For each stage i we define A_i to be the set of operations that have already been scheduled up to stage i. Initially $A_0 = \emptyset$. In the first stage ($i = 1$), we schedule at $t = 0$ all the operations that have indegree 0. Because DC is acyclic, at least one operation with indegree 0 exists. At stage i we schedule all the operations that have indegree 0 in $DC - A_{i-1}$. An operation (i,j) will start at $st_{ij} = \max\{ct_{kl} \mid (k,l) \rightarrow (i,j)\}$. The uniqueness is forced by semi-activeness; i.e., each operation is scheduled as early as possible while keeping the order defined by DC. •

Corollary 1 *Solving* $O|pconc|C_{max}$ *is equivalent to the problem of orienting the edges of the conflict graph so that the digraph obtained will be acyclic and the maximally-weighted path will be minimal.*

The weight of a path is the sum of all the processing times of operations (vertices) in that path.

Example 2 A PCOSS instance is composed of a matrix PT and a graph CC, or CC. We take PT of Example 1 and CC of Figure 2(b). A possible schedule for this PCOSS instance is shown in Figure 3 by its Gantt chart and the corresponding orientation of the conflict graph, i.e., the partial-sequence graph DC.

The rank matrix representation follows. We define the rank r_{ij} of an operation (i, j) as the number of vertices in a longest path in DC from a source vertex to the operation (i, j). The rank matrix is denoted $R = [r_{ij}]$.

The rank matrix of Example 2 is

$$R = \begin{bmatrix} 1 & 2 & 4 & 1 \\ 1 & 3 & 1 & 3 \\ 1 & 4 & 1 & 3 & 2 \end{bmatrix} \quad (3)$$

The rank matrix R has the following two properties:

1. $r_{ij} \neq r_{kl}$ for each conflict pair (i,j) and (k,l).

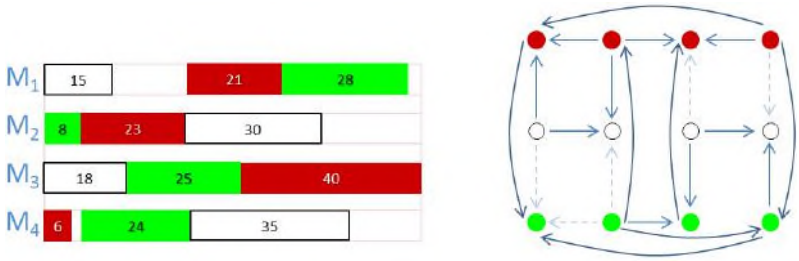

Fig. 3 A possible PCOSS schedule is given by its (a) Gantt chart, or its corresponding (b) partial-sequence graph. Dashed arcs can be removed due to transitive closure.

2. For any operation (i, j) with $r_{ij} > 1$ there exists a conflicting operation (k, l) with $r_{kl} = r_{ij} - 1$.

Given a concurrence graph for a PCOSS instance, we call a positive integer-valued matrix nxm a *CG-rectangle* if it satisfies conditions 1 and 2. In the case of a standard OSS the concurrence graph has only edges connecting operations if they belong to different jobs and different machines. The CG-rectangle then reduces to a Latin-rectangle. The following theorem generalizes the "Latin-rectangle theorem" [7].

Theorem 1 *An* n *x* m *matrix* $A = [a_{ij}]$ *is a CG-rectangle iff it is a rank matrix of a semi-active schedule for a PCOSS problem with* n *jobs and* m *machines.*

Proof Given a semi-active schedule for a PCOSS problem, a rank matrix is constructed as defined previously. On the other hand, given a CG-rectangle, A, we construct an appropriate partial-sequence graph by orienting the edges of the conflict graph as follows: an edge $\{(i, j), (k, l)\}$ will be oriented $(i, j) \rightarrow (k, l)$ if $a_{ij} < a_{kl}$ and $(k,l) \rightarrow (i, j)$ if $a_{kl} < a_{ij}$. The obtained graph is obviously acyclic. The theorem's assertion then follows from Lemma 1. •

3 Complexity issues

Concerning the complexity of $O|pconc|C_{max}$, it is proved next that the problem with only one job and unitary processing times, denoted $O|pconc, n = 1, p_{ij} = 1|C_{max}$, is already NP-Hard. It immediately follows that a general PCOSS is also NP-Hard. It is worth noting that with one job, minimizing the makespan in both the standard OSS problem and the concurrent open shop is a trivial task. For $O|n = 1|C_{max}$, any order of the operations leads to a schedule with $C_{max} = \sum_{j=1}^{m} p_{1j}$. For $O|conc, n = 1|C_{max}$, any order of the operations leads to a schedule with $C_{max} = \max_j\{p_{1j} \mid 1 < j < m\}$. Moreover, for the standard open shop problem with unitary processing times, Bräsel et al. developed a polynomial-time algorithm for minimizing the makespan for *any number* of jobs [9].

Theorem 2 *The problem* $O|pconc, n = 1, p_{ig} = 1|C_{max}$ *is NP-Hard.*

Proof In the case where there is only one job and all processing times equal 1, the makespan of any schedule S is the length of the longest path in the sequence graph defined by S, which is the maximum rank in the matrix defined by S. By Corollary 1 the opposite is also true – the length of the longest path in any acyclic orientation of the conflict graph is the makespan of a feasible schedule for the problem. Because the conflict graph can be any arbitrary undirected graph on the set of the n operations, it follows that $O|pconc, n = 1, p_{ig} = 1|C_{max}$ is equivalent to the problem of orienting an undirected graph in order to minimize the size of the longest directed path. The latter is polynomially equivalent to the Graph-Colouring problem by the proof of the Gallai-Roy-Vitaver theorem [3], which asserts that the minimal size of the longest directed path in an orientation of an undirected graph G is equal to the chromatic number of G. Because the Graph-Colouring problem is NP-Hard it follows that $O|pconc, n = 1, p_{ig} = 1|C_{max}$ is NP-Hard. •

Corollary 2 *The general problem* $O|pconc|C_{max}$ *is NP-Hard.*

4 Constructive heuristic

The constructive heuristic considered in this section is an adaptation to the partially-concurrent case of the insertion algorithm that was proposed for standard OSS [10]. The algorithm builds a full schedule (rank matrix) in an iterative manner. At each iteration, the algorithm *inserts* one operation into a partial schedule, until a full schedule is reached. The order at which the operations are inserted is determined before the iterative process commences.

Following [10], the operation insertions are combined with *beam search*. That is, only a limited number of solution paths within the complete search space are investigated. During the search process, each partial schedule has one *parent* and several *children*. The parent is the partial schedule, excluding the last inserted operation. The children result from the insertion of the next operation. The number of parallel solution paths is limited by the beamwidth p.

We consider the following possibilities for the insertion of an operation (i, j) into the partial rank matrix R:

1. $r_{ig} = 1$. This means that i is the first job in the job order on machine j and j is (one of) the first machine(s) in the machine order of job i.
2. $r_{ig} = r_{kg} + 1$, where r_{kg} is any of the values that appear in column j of R. This means that operation (i, j) becomes a direct successor of one of the operations on machine j.
3. $r_{ig} = r_{il} + 1$, where r_{il} is any of the values that appear in row i of R that correspond to an operation of J_i, which is in conflict with operation (i, j). This means that operation (i, j) becomes a direct successor of one of the operations of job i, with which it has conflict.

These possibilities correspond to the cases (c1) and (c2) in the original insertion algorithm [10].

An insertion of r_{ij} to R results in a new (partial or full) matrix $\boldsymbol{R'}$. However, R' may possibly not be a CG-rectangle. This happens when one or more of the following conflicts occur: (I) $k = 6_i \bullet [r_{ij} = r_{kj}]$; (II) $l = 6_j \bullet [r_{ij} = r_{il}$ and (i,j) conflicts with $(i,l)]$. Note that there may possibly be several instances of conflict (II) due to concurrent machines. All the conflicts are resolved by incrementing by 1 all the conflicting entries. The entries with incremented values may in turn be in conflict with a new set of entries. This process continues until all conflicts in R' are resolved.

The set of all obtained R' matrices (each corresponding to a possible value of r_{ij}) forms the list of children. In each iteration of the beam search, a list of the most promising p children should be selected. Following [10], we consider two variants:

– **INSERT1:** In each iteration we select the p-best children from the whole set. This means that some selected children may have the same parent.
– **INSERT2:** For each of the p parents we select the best child. This means that all children have different parents. The first variant is applied as long as we do not have p parents.

We still must decide which children are considered the best in each step. Similarly to [10], we assign to each child the cost of the longest path (cost-wise) that goes through the newly inserted operation (i, j) in the relevant R' matrix. A path in PCOSS is a series of adjacent operations in the conflict graph CG. The children that are considered best are those with lowest costs. A more detailed description of the insertion algorithm can be found in [10].

5 Experimental evaluation

The objective of the experimental evaluation of the present paper is twofold. Naturally, one would like to evaluate the effectiveness of the proposed constructive heuristic in terms of the quality of the obtained solutions. Nevertheless, perhaps even more interesting is the question of how partial concurrency affects the problems themselves, regardless of the chosen solution method.

In an attempt to shed light on these two issues, we turned to the commonly used OSS problem instances that were proposed by Taillard [20]. We used the entire set of Taillard's OSS benchmark that consists of six problem sizes (4x4, 5x5, 7x7, 10x10, 15x15, 20x20), with 10 instances of each problem size. For each of these 60 standard OSS problems we created 90 new PCOSS instances with varying concurrency levels, where the concurrency level of a problem relates to the percentage of non-conflicting operation pairs out of all operation pairs sharing the same job. We used concurrency levels with 10% intervals, varying from 10% to 90%. For each concurrency level we generated 10 problem instances, each with a randomly chosen set of non-conflicting operation pairs. For each problem in Taillard's benchmark, we also considered the two PCOSS

extremes, which are the original (standard OSS) problem (0% concurrency) and the fully-concurrent problem (100% concurrency).

Following [10], we consider three versions of the constructive heuristic: one with beamwidth $p = 1$, and the two INSERT variants with beamwidth $p = 2$. In order to evaluate the quality of the obtained solutions we relate to the deviation of the corresponding C_{max} from the machine lower bound. For each problem, the machine lower bound is the maximal working time of any of the machines, given by $\max\{\sum_j p_{ij} | 1 < j < m\}$.

Figure 4 presents the quality of obtained solutions for the PCOSS problems of size 4x4. The results shown for each concurrency level are the average deviations of all the PCOSS problems of that level. The results in Figure 4 reveal interesting information regarding both issues of the evaluation's objective. First, the obtained solutions of the heuristic are clearly of high quality, even when using a very low beamwidth. These findings coincide with the respective results for standard OSS [10][1]. Second, the graph shows that as the concurrency level increases the optimal C_{max} gets closer to the machine lower bound. This is not surprising, since more concurrency brings with it more efficient scheduling possibilities that in turn reduce C_{max} towards the bound set by the machines.

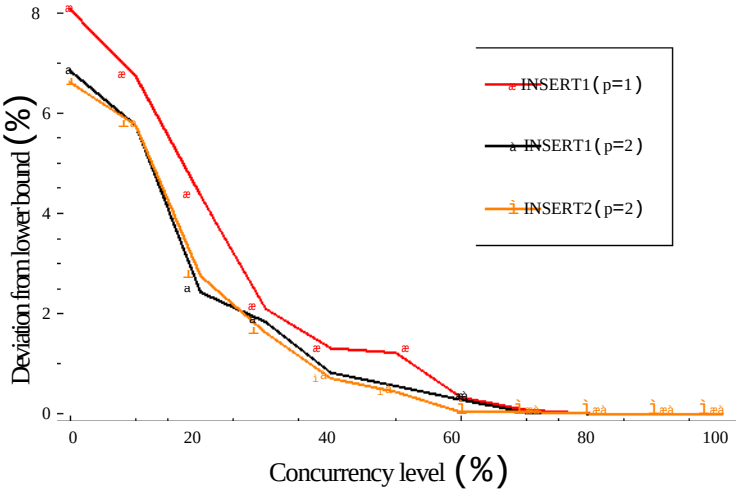

Fig. 4 The average deviation of C_{max} from the machine lower bound as a function of the concurrency level for problems of size 4×4.

For high levels of concurrency the heuristic algorithm gave the optimal solution. Therefore, in order to evaluate the scalability of the constructive heuristic we focus on a low concurrency level (10%). Figure 5 presents the quality of obtained solutions for PCOSS problems of different sizes. The displayed results

1 The results for 0% concurrency are worse than those presented in [10], since the machine lower bound used herein is looser than the (machine and job) lower bound used in [10].

are the average deviations of 100 PCOSS problems of each problem size, with the size ranging from 4 × 4 to 20 × 20 according to the Taillard benchmark. Again, the obtained solutions of the heuristic are of very high quality. The results also indicate that as the problems get larger the optimal C_{max} gets closer to the lower bound. A similar phenomenon was observed for standard OSS problems [10].

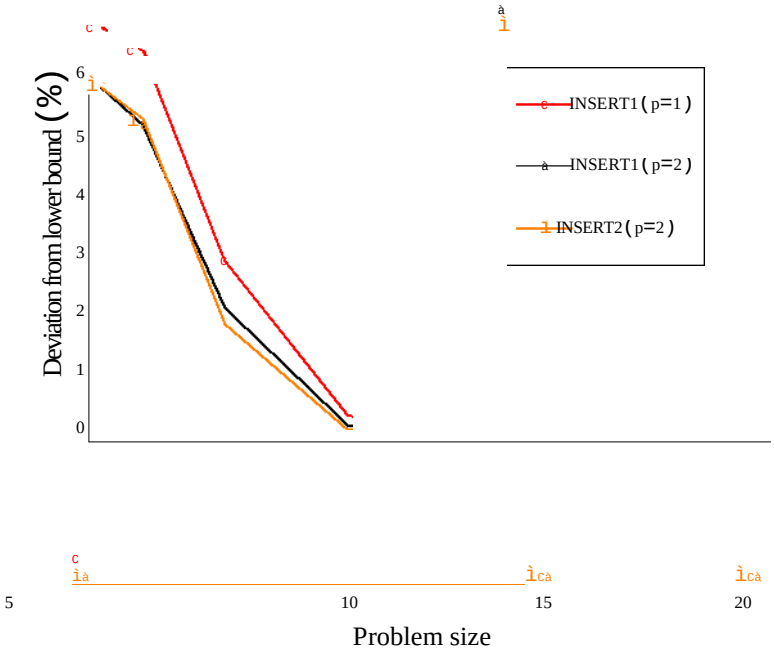

Fig. 5 The average deviation of C_{max} from the machine lower bound as a function of the problem size ($n \times n$) for problems with 10% concurrency level.

Another aspect is the runtime of the constructive heuristic. The advances in hardware capabilities enable us to run the above experiments in reasonable time. In fact, even the hardest considered PCOSS problems (size 20 × 20) were solved in less than half a second with beamwidth $p = 2$ on a hardware comprised of Intel i5 4th generation and 8GB memory.

6 Discussion

The presented PCOSS is a general open shop problem, connecting two previously discussed scheduling problems – that of the standard open shop and its concurrent version. PCOSS enables natural representation of various realistic problems, such as that of the airplane garage that was mentioned in the introduction section. By extending the rank matrix scheme to the more general scenario of PCOSS, one may utilize efficient solving techniques, such as the presented constructive heuristic.

Investigating this general problem has highlighted the fact that generating an open shop schedule (whether concurrent or not) is equivalent to orienting a conflict graph, i.e., generating an acyclic digraph DG. The digraphs formally

presented in the literature with respect to the non-concurrent OSS are basically transitive reductions of the partial-sequence graphs presented herein. Existing studies on the topic of generating acyclic orientations of a given undirected graph [4] can shed light on problems of OSS.

In this article we assumed that jobs in a PCOSS can be split to be processed on several machines simultaneously, but each machine hosts only one job at a time. Consequently, all the vertical edges existed in the conflict graph. Our proposed formalism enables the removal of this limitation, allowing also some operations of a given machine to be processed concurrently. The obtained rank matrix might then have several entries on a given column that are the same, as long as the corresponding operations are not conflicting.

We have also assumed that $p_{ij} > 0$ for all $i \in I$ and $j \in J$, i.e., that all the vertices $I \times J$ represent real operations, with each job visiting all the machines. In real life it often happens that some jobs visit only a partial set of machines. It is possible to include these scenarios without changing the given formalism: $I \times J$ vertices are considered as proposed before. A non-processing operation, i.e., any operation of a job i that does not visit machine j, has zero processing time. A vertex (i, j), which represent a non-processing operation has concurrent edges to all the other vertices, i.e., in the conflict graph it is not connected to any other vertex. The rank of non-processing operations will then be $r_{ij} = 1$, because it has 0 indegree and is therefore, by definition, a source. Note that this procedure can lead to several 1's in a given column. Nevertheless, the rank matrix remains a CG-rectangle.

PCOSS was shown to be NP-Hard. Yet, the heuristic algorithm proposed in Section 4 reaches the optimum in many instances, even when using a narrow beam. The examined instances are that of Taillard, with varying concurrency levels. For these instances, increasing the concurrency level resulted in obtaining C_{max} values that are very close to the machine lower bound. These results can be misleading, suggesting that increasing the concurrency level makes the problem easier to solve. Taillard's benchmark is a standard for non-concurrent OSS, for which the most difficult problems are those of square PT, with $n = m$. Contrary to that, the possibility of processing a given job in several machines concurrently suggests that it might be harder to solve problems with $m > n$. The logic behind this assertion is demonstrated by considering a uniform $n \times n$ OSS instance (uniform in the sense that $p_{ij} = 1$ for all i and j). This is an easy OSS problem – a schedule represented by any Latin rectangle $LR(n, n, n)$ is optimal with $C_{max} = n$. A uniform OSS problem with $m > n$ *is* still easy – an optimal schedule is given by $LR(n, m, m)$, with $C_{max} = m$. However, in uniform PCOSS problems things are more complicated. The squared problem remains easy, because C_{max} cannot be lower than the machine lower bound. Nevertheless, a uniform PCOSS with $m > n$ *is* completely non-trivial. Even obtaining an appropriate job lower bound is NP-Hard, being equivalent to the maximum independent set problem.

Further generalizations can be achieved considering PCOSS and its representation. For example, it is possible to extend the definition of a reducible sequence given in irreducibility theory [2,8]. The known benchmarks should

be extended to include PCOSS instances that describe the schedule difficulties more appropriately. We leave these interesting issues for future research.

Another interesting direction for future work is to model and solve the most general scenario of an airplane garage, which was the initial motivation for the present study. In the general real-life problem the airplanes are located in several hangars, and some of the tasks need teams of technicians. We plan to additionally generalize the OSS model to include issues of technicians' transportation and teaming, which were previously referred to in *Workforce Scheduling and Routing Problems [11].* Additionally, some real-life scenarios are *multi-mode* (cf. [13]), i.e., some tasks may be performed in several ways (modes) using different amounts of resources (technicians). The adaption of PCOSS to enable multiple modes is another challenging prospect.

References

1. Andresen, M., Br¨asel, H., Mörig, M., Tusch, J., Werner, F., Willenius, P.: Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. Mathematical and Computer Modelling 48(7), 1279–1293 (2008)
2. Andresen, M., Dhamala, T.N.: New algorithms and complexity status of the reducibility problem of sequences in open shop scheduling minimizing the makespan. Annals of Operations Research 196(1), 1–26 (2012)
3. Bang-Jensen, J., Gutin, G.: Theory, algorithms and applications. Springer Monographs in Mathematics, Springer-Verlag London Ltd., London (2007)
4. Barbosa, V.C., Szwarcfiter, J.L.: Generating all the acyclic orientations of an undirected graph. Information Processing Letters 72(1), 71–74 (1999)
5. Br¨asel, H.: Matrices in shop scheduling problems. In: Perspectives on Operations Research, pp. 17–41. Springer (2006)
6. Brasel, H., Harborth, M., Willenius, P.: Isomorphism for digraphs and sequences of shop scheduling problems. Journal of combinatorial mathematics and combinatorial computing 37, 115–128 (2001)
7. Br¨asel, H., Kleinau, M.: On the number of feasible schedules of the open-shop-problem-an application of special latin rectangles. Optimization 23(3), 251–260 (1992)
8. Br¨asel, H., Kleinau, M.: New steps in the amazing world of sequences and schedules. Mathematical methods of operations research 43(2), 195–214 (1996)
9. Br¨asel, H., Kluge, D., Werner, F.: A polynomial algorithm for an open shop problem with unit processing times and tree constraints. Discrete Applied Mathematics 59(1), 11–21 (1995)
10. Brāsel, H., Tautenhahn, T., Werner, F.: Constructive heuristic algorithms for the open shop problem. Computing 51(2), 95–110 (1993)
11. Castillo-Salazar, J.A., Landa-Silva, D., Qu, R.: A survey on workforce scheduling and routing problems. In: Proceedings of the 9th international conference on the practice and theory of automated timetabling, pp. 283–302 (2012)
12. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics. v5 pp. 287–326 (1977)
13. Kolisch, R., Drexl, A.: Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE transactions 29(11), 987–999 (1997)
14. Leung, J.Y.T., Li, H., Pinedo, M.: Order scheduling in an environment with dedicated resources in parallel. Journal of Scheduling 8(5), 355–386 (2005)
15. Mastrolilli, M., Queyranne, M., Schulz, A.S., Svensson, O., Uhan, N.A.: Minimizing the sum of weighted completion times in a concurrent open shop. Operations Research Letters 38(5), 390–395 (2010)

16. Naderi, B., Fatemi Ghomi, S., Aminnayeri, M., Zandieh, M.: A contribution and new heuristics for open shop scheduling. Computers & Operations Research 37(1), 213–221 (2010)
17. Ng, C., Cheng, T.C.E., Yuan, J.: Concurrent open shop scheduling to minimize the weighted number of tardy jobs. Journal of Scheduling 6(4), 405–412 (2003)
18. Pinedo, M.: Scheduling: theory, algorithms, and systems. Springer (2012)
19. Roemer, T.A.: A note on the complexity of the concurrent open shop problem. Journal of scheduling 9(4), 389–396 (2006)
20. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64(2), 278–285 (1993)
21. Wagneur, E., Sriskandarajah, C.: Openshops with jobs overlap. European Journal of Operational Research 71(3), 366–378 (1993)