An iterated greedy algorithm for the partial job shop scheduling problem

Tadeu Zubaran and Marcus Ritt
Instituto do Informática
Universidade Federal do Rio Grande do Sul, Brazil
email: {tkzubaran,marcus.ritt}@inf.ufrgs.br

***Abstract*—The open and job shop scheduling problems are widely studied in the literature. In this paper we study the partial job shop scheduling problem proposed by Nasiri and Kianfar [14]. In this problem, the operations of a job are partially ordered. An open shop and a job shop are therefore special cases of a partial shop, with no order or a linear order of the operations, respectively. We propose an iterated greedy heuristic based on an extension of the well-known shifting bottleneck heuristic for the job shop to partial shops. In computational experiments we evaluate the performance of these heuristics and compare it to results from the literature.**

Keywords. Partial Shop scheduling, Heuristics, Shifting Bottleneck, Iterated Greedy Algorithm

I. Introduction

Job shops and open shops are well-known models for production processes. In the *job shop scheduling problem* (JSSP) and *open shop scheduling problem* (OSSP) we have to find an optimal schedule for a set of jobs $J = [n]$ on a set of machines $M = [m]$[1]. The set of operations $O = \{o_{ij} \mid (i,j) \in M \times J\}$ contains one operation for each machine-job pair. Each operation $o_{ij} \in O$ has a processing time of p_{ij}. Each machine can process only one job at a time, and each job can be processed only on one machine at any instant. Once an operation is scheduled it must be processed to completion, so no preemption is allowed. We consider no explicit setup times, since without preemption, we can assume that the setup time is included in the processing time. We can model processes that have jobs which do not use a particular machine by setting the time of its operations to zero.

What differentiates JSSP and OSSP is the precedence order of the operations of each job. In the JSSP each job visits the machines in a particular total order, while in the OSSP we can execute the operations of a job in any order.

The literature contains studies focusing on different objective functions, e.g. total flow time, or total lateness. In this paper we focus on the most common objective: to minimize the *makespan* $C_{\max}$, i.e. the time needed to complete all jobs.

In this work we study an extension of the JSSP where the order of the operations of the jobs is partial instead of linear, the so called *Partial Shop Scheduling Problem* (PSSP), proposed by Nasiri and Kianfar [14]. The OSSP is an special case of PSSP where the partial order is empty, while the JSSP is a special case of PSSP where the partial order is total. The *order strength* of a partial order is the fraction of actual precedence relations t of all possible precedence relations, defined as $\text{OS} = t/\binom{n}{2}$. Instances of the JSSP have order strength 1, and instances of the OSSP order strength 0. Since both OSSP and JSSP are NP-hard [17] the PSSP also is.

[1]We use the notation $[n] = \{1, \ldots, n\}$.

A. Literature Review

The JSSP is well known to be a hard combinatorial optimization problem. The notorious instance with ten machines and jobs proposed by Fisher and Thompson [9] was solved only twenty years later by a branch-and-bound algorithm by Carlier and Pinson [7]. Other noteworthy exact solvers are Applegate and Cook [2] and Brucker et al. [5], which are branch-and-bound algorithms as well.

To the best of our knowledge no exact solver is able to consistently solve instances with twenty machines and jobs or more in a timely manner, and a growing effort has been put into efficient heuristic solvers. Nowicki and Smutnicki proposed two Tabu search algorithms [15, 16]. Watson et al. [23] highlighted the importance of the neighbourhood in the efficiency of the Tabu search.

Adams et al. [1] created the Shifting Bottleneck Procedure, which is a fast constructive heuristic. Balas and Vazacopoulos [3] and Pezzella and Merelli [18] utilized the algorithm within a guided local search and a Tabu search, respectively.

Most of the early works on the OSSP are studies on complexity results and polynomial solutions or approximations for special cases. Gonzalez and Sahni [10] proposed a linear time exact solver for the special case with two machines, and proved NP-hardness of the case with more than two machines. Chen and Strusevich [8] present approximation algorithms for the case with three machines.

From more recent works we cite the branch and bound by Brucker et al. [4] and the genetic algorithm by Prins [19]. Ramudhin and Marier [20] extended the Shifting Bottleneck Procedure for general shops.

The *Mixed shop scheduling problem* was proposed by Masuda et al. [12]. The jobs in a mixed shop have either the structure of an OSSP or of a JSSP. The jobs are divided into sets J_o and J_j ($J_j \cup J_o = J$) which is the set of OSSP type jobs and JSSP type jobs respectively.

The *Stage shop scheduling problem* proposed by Nasiri and Kianfar [13] is an extension of the mixed shop. The jobs have stages and the stages contain operations. The instance defines a total order between the stages while the operations in a stage are independent. An operation may only be executed if all operations of all previous stages were finished. Operations of the same stage may be executed in any order.

Nasiri and Kianfar [14] formalized the PSSP, proposed a mixed integer formulation, implemented a hybrid genetic algorithm, and proposed a of set instances for the problem based on the instances of the JSSP by Taillard [22]. We use the instances proposed by this paper to evaluate the performance of our algorithms and compare to their results.

The remainder of this paper is organized as follows. In Section II we define PSSP formally, in Section III we explain the Shifting Bottleneck procedure proposed by Adams et al. [1], and present an extension to the PSSP in Section IV, and an iterated greedy algorithm in Section V. In Section VI we present the results of computational tests with these methods. We conclude in Section VII.

Algorithm 1 Shifting bottleneck procedure

```
$M_0 = \emptyset$
**while** $M_0 \neq M$ **do**
    Identify the bottleneck machine $m_b \in M \setminus M_0$
    Schedule $m_b$ optimally
    $M_0 = M_0 \cup \{m_b\}$
    **while** 3 times, or until no improvement is made **do**
        **for** each machine $m \in M_0$ **do**
            Re-optimize $m$
        **end for**
    **end while**
**end while**
**while** Improvement is made **do**
    **for** each machine $m$ **do**
        Re-optimize $m$
    **end for**
**end while**
```

II. Problem Definition

A. Digraph Representation

We can represent an instance of a shop scheduling problem and its solution in the form of a disjunctive graph $G = (V, C, D)$ with three components:

- A set of nodes V. Each operation of the instance is represented by a node in the graph. Each node has an associated cost p_{ij} which is the processing time of job j on machine i. In addition we have two special nodes, the source and the sink, which have no cost and are associated with no machine.
- a set of directed conjunctive arcs C. Any precedence between operations is represented by these arcs. In addition we insert a conjunctive arc going from the source to each operation without any predecessor, and symmetrically one from each operation without any successor to the sink.
- a set of undirected disjunctive arcs D. For every pair of nodes which are not connected directly or transitively we add a disjunctive arc connecting both.

The scheduling is reduced to an orientation of all disjunctive arcs. The fact that two nodes are not connected by the elements of C (and the transitive closure them) means that no order between the operations is defined by the instance. Therefore we can select their order arbitrarily. A complete valid schedule in the graph model is a *complete selection*. A complete selection is an orientation of D such that each element of D is oriented and the graph is acyclic.

Given a disjunctive graph and an acyclic selection we can derive a valid schedule by starting the operations at the time given by the longest path from source to the corresponding node in V. The makespan of such a schedule is the longest path between source and sink. A *critical path* is a longest path from the sink to the node.

The JSSP, the OSSP, and the PSSP differ only in the set of conjunctive and disjunctive arcs. In the JSSP, C contains all the precedences associated with the total order of each job, and D contains all the machine cliques. An acyclic selection of a machine clique is equivalent to selecting a processing order for the operations of that machine. Therefore a complete selection corresponds to a processing order for each individual machine. In the OSSP C is empty, and D contains all the arcs corresponding to the machine and job cliques. A complete selection is equivalent to an ordering of all the operations in each job and machine. Finally, in the PSSP, C corresponds to the partial order relation of each job, while D contains the machine cliques and the arcs between operations with undefined order in each job. A complete selection is equivalent to an ordering of the operations in each machine and choosing a complete ordering for each job that satisfies the partial ordering defined by the instance.

III. The Shifting Bottleneck Procedure

The *Shifting Bottleneck Procedure* (SBP) proposed by Adams et al. [1] is a constructive heuristic for the JSSP.

Algorithm 1 shows the pseudocode for the SBP. The algorithm has two main steps:

1) Identify and schedule the bottleneck machine.
2) Locally re-optimize already scheduled machines.

The set M_0 contains all machines which were already scheduled. The first step selects a new bottleneck machine, adds it to M_0 and finds the optimal acyclic selection of the corresponding one-machine problem. The re-optimization step revisits the machines already in M_0 scheduling each one optimally again.

Both steps utilize the one-machine problem with release and delivery times associated with each single machine. Given a partial orientation of D in G we can schedule a single machine $i \in M$, i.e., select an orientation of the arcs in D of the machine clique corresponding to i. The length of the longest path from the source to an operation is called is *head*. Similarly, the length of the longest path from an operation to the sink is its *tail*. An operation of the one-machine problem under consideration cannot start before its head, and consumes a extra time corresponding to its tail after completion. Therefore the corresponding one-machine problem defines, for an operation o, a release time r_o equal to its head, and a delivery time q_o equal to its tail.

The bottleneck machine m_b is the machine with the highest makespan for the associated one-machine problem. The SBP utilizes the branch-and-bound solver for the one-machine problem proposed by Carlier [6] to find an optimal schedule. Notice that it is possible to have an acyclic selection for a machine which yields a cycle in the overall graph. When G contains a cycle the schedule is undefined. To solve this problem after optimizing a machine we identify cycles, and, if needed, we perform the branch-and-bound procedure again enforcing inverse precedences of the ones causing the cycles.

IV. A SHIFTING BOTTLENECK PROCEDURE FOR THE PSSP

In the JSSP we need to establish an ordering for the operations on each machine. For the PSSP we need to do so for both the machines and the jobs. Thus either machines or jobs can be bottlenecks, and are subject to an optimization of the order of its operations.

We create a one-machine problem for each job of the instance the same way we do for the machines, therefore we define the release and delivery times as the heads and tails in the current schedule. Different from machine bottlenecks, we must force the branch-and-bound algorithm to consider only solutions that satisfy the partial order of the operations defined by the instance.

A. Implementation Details

We can compute the maximum distance from the source and to the sink for any operation of the instance in linear time in the number of operations the instance contains. This happens because the transitive arcs of G can be ignored: a longest path never uses them. Thus we can compute the maximum distance by updating it in topological order. Notice that we need to perform the update for each operation in the instance at each step.

When we solve the one-machine problem and select an order for the operations of a job or a machine it is possible that a cycle in G is created. Assume we are optimizing a machine or job e and the branch-and-bound determines that in the optimal schedule for e operation o_1 precedes (directly or indirectly) operation o_2. Let G_e be equal to G without the disjunctive arcs associated with e. If there is a path from o_2 to o_1 in G_e then scheduling o_1 before o_2 will form a cycle and the branch-and-bound algorithm will generate an invalid schedule. Because forming cycles is rare we do not test for possible cycles during optimization. After the branch-and-bound algorithm terminates, when we compute the new heads and tails of the instance, we can detect the existence of a cycle. Only in the cases where we find one or more cycles we compute the set of precedences P from e which are part of a cycle. We solve the branch-and-bound again for e but subject to the inverse precedences of the elements of P.

Algorithm 2 Schrage Schedule

```
$U$ is the set operations to be scheduled.
$t = \min_{o \in U} r_o$
while $U \neq \emptyset$ do
    $R = \{o \mid o \in U \wedge r_o \leq t\}$
    $o_s = \text{argmax}_{o \in R}(q_o)$
    Remove $o_s$ from $U$ and schedule it at $t$
    $t = \max(t + p_o, \min_{o \in U} r_o)$
end while
```

During all stages of the algorithm we keep G acyclic, so when we define the order of the operations for a job or machine e, for each pair of operations o_1 and o_2 of e there is always a possible order that does not create a cycle, otherwise o_1 would have a path to o_2 and o_2 to o_1 in G_e which means G would have a cycle before we schedule e.

During the computation of the heads and tails we only need to know the immediate predecessors and successors of an operation G. Yet to detect and fix cycles and to enforce the partial orders we need the transitive precedences as well, so we keep a data structure for the transitive closure of the precedence orders as proposed by Italiano [11], so the verification of a precedence is done in constant time.

Our implementation of Carlier's one-machine branch-and-bound algorithm must be able to enforce the precedences in a partial order relation, either to avoid cycles or to obey the partial order associated with a job. The algorithm is based on Schrage's schedule described in Algorithm 2, which at a given time schedules the released operation with the longest delivery time.

To enforce the precedences we adjust the release dates and delivery times of the one-machine problem. For each operation o_1 that precedes an operation o_2 we set the release dates and delivery times as follows:

$$r_\beta = \max(r_\beta, r_\alpha + d_\alpha) \tag{1a}$$

$$q_\alpha = \max(q_\alpha + q_\beta + d_\beta), \tag{1b}$$

Equation 1a guarantees that o_2 will only be ready once o_1 is, and Equation 1b will give o_1 priority in the Schrage schedule, such that it will be scheduled before o_2. Since o_1 is always scheduled before o_2 there is no possibility that the increased delivery time of o_1 will change the makespan of the schedule. We adjust the release dates and delivery time in this fashion for each precedence in the partial order. We adjust the heads of each operation only according to its direct successor and predecessor in the topological order, such that the transitive relations will be enforced naturally.

V. AN ITERATED GREEDY ALGORITHM FOR THE PSSP

An *iterated greedy algorithm* is a constructive meta-heuristic introduced by Ruiz and Stützle [21]. A general iterated greedy algorithm is described in Algorithm 3. It first builds an initial solution and then repeatedly perturbs the current solution and applies a local search to the newly generated solution. Different from an iterated local search, the

```
Algorithm 3 Iterated Greedy Algorithm
  Generate a initial solution
  while time is not up do
      Destroy part of the solution
      Rebuild solution
      Apply local search
      Test acceptance of the new solution
  end while
```

perturbation is realized by destruction and construction. In the destruction step, some random elements are removed from the solution. These elements are inserted greedily into the solution in the construction step, to obtain a new complete solution.

A. *Initial Solution*

To produce a good initial solution we utilize SBP adapted for the PSSP as explained in the previous section.

B. *Perturbation*

After each step the current solution is always a local minimum as left by the local search (explained next section). In order to escape local minima we partially destroy the current solution and perform a greedy reconstruction. This perturbation depends on a parameter R, which indicates how many jobs or machines will be removed from and reinserted greedily into the current solution. We randomly select R elements to be removed, which can be any combination of jobs and machines, and remove the disjunctive arcs associated with them. We then reinsert these elements in a random order. For each reinsertion we compute the heads and tails associated with the element and solve the one-machine problem optimally with Carlier's algorithm as described above.

C. *Local Search*

To improve the quality of the new solutions generated by the perturbation we perform a first improvement local search. We can generate a new solution for a JSSP instance by swapping the order of two operations on the same machine. This is equivalent to inverting the orientation of the disjunctive arcs between the two operations in G. To be able to improve the makespan we have to swap two operations on some critical path. Given a critical path, sequences of three or more operations on the same machine are called blocks. We call the first and last operations of a block the edges. Nowicki and Smutnicki [16] proposed a neighbourhood where each element is generated by swapping a block edge with the adjacent operation in the same block. We adapted this neighbourhood for PSSP by extending the notion of blocks to *machine blocks* and *job blocks*, which are sequences of three or more operations from the same machine or job, respectively.

We define the neighbourhood of a solution in our local search as all solutions generated by swapping one operation at the edge of a block with an adjacent element in the same block. In case of a job block we check if the swap is allowed by the partial order of the job and discard the neighbour if the swap is invalid.

Table I

RESULTS OF THE PARAMETER SETTING: AVERAGE RPD OVER 50 INSTANCES FOR EACH COMBINATION OF THE PARAMETERS R AND α.

ARPD	α		
R	0.40	0.50	60
1	22.53	22.76	22.39
2	5.46	5.25	5.72
3	**5.01**	5.60	5.73
4	5.24	5.91	6.70

D. *Acceptance Criterion*

To improve the diversity of the search we allow it to decrease the quality of the current solution. To do so we utilize a *Metropolis* acceptance criterion. If the new solution is better than the current one it is always accepted. If its makespan is Δ units worse we accept it with probability

$$e^{-\Delta/T} \tag{2}$$

where T is a parameter, the so-called temperature. We define the temperature of an instance by

$$T = \alpha \frac{p_{mean}}{10} \tag{3}$$

for a parameter α and an average processing time of

$$p_{mean} = \frac{\sum_{j \in J} \sum_{m \in M} p_{ij}}{nm}. \tag{4}$$

VI. COMPUTATIONAL EXPERIMENTS

In this section we evaluate the effectiveness of our iterated greedy algorithm for the PSSP and compare it to the results of the hybrid scatter search proposed by Nasiri and Kianfar [14]. The test instances have been generated in the same as those of Nasiri and Kianfar [14]. They are based on the first 50 of the 80 instances for the JSSP proposed by Taillard [22]. For each job shop instance, a partial shop instance is created by removing all precedences. Then, for each job j we insert $\eta_j \in (0, m)$ random precedences, without forming cycles. For each job shop instance 10 random instances are created this way, such that the total test suite contains 500 instances.

We implemented our iterated greedy algorithm for the PSSP in C++ and executed the experiments on a PC with an AMD FX 8150 processor with eight cores running at 2.80 GHz and 12 GB of main memory. Each test used only one core.

A. *Parameter Adjustment*

To calibrate the parameters α and R we executed our algorithm one of the ten instances based on the 50 job shop instances. We executed the algorithm for each combination of $\alpha \in \{0.4, 0.5, 0.6\}$ and $R \in \{1, 2, 3, 4\}$. Nasiri and Kianfar [14] execute each instance for a variable amount of time. We executed the algorithm for each instance with the same time limit to adjust the two parameters.

We measure the quality of a solution by its relative percentage deviation (RPD) from a lower bound L

$$RPD = 100 \frac{C_{\max} - L}{L} \%, \tag{5}$$

and use the machine lower bound

$$L = \max_{i \in M}(\min_{j \in J}(h_{ij}) + \sum_{j \in J} p_{ij} + \min_{j \in J}(t_{ij})), \tag{6}$$

where h_{ij} is the head of operation o_{ij} and t_{ij} its tail.

In Table I we report the average RPD over all 50 instances for all parameters settings. The algorithm seems to be robust for different parameter settings, except for the case $R = 1$ which produces substantially worse results. In the remaining experiments we utilize the parameter setting $R = 3$ and $\alpha = 0.4$ since it yields the best results.

B. *Results for the iterated greedy algorithm*

In this section we compare the results of our iterated greedy algorithm to those of the hybrid scatter search of Nasiri and Kianfar [14]. Nasiri and Kianfar [14] used an PC with an Intel Core Duo T2400 processor running at 1.83 GHz, which is about a factor 2.4 slower than our machine. Their algorithm runs until a termination criterion is satisfied. For a fair comparison we run our algorithm with a time limit equal to the median time of each set of ten instances based on the same job shop instance, corrected by the above factor.

The results of the computational experiments are shown in Table II. For each instance we report the number of jobs (column "n") and the number of machines (column "m"), and the average RPD for each group of ten instances and the total execution time in seconds for the hybrid scatter search (columns "HSS") and our method (columns "IGA").

The iterated greedy algorithm obtains a better average RPD than the hybrid scatter search in 33 of the 50 instance groups. In particular for the 30 instance groups with up to 20 jobs, it is better in 23 cases. For the 20 larger instance groups, both algorithms have a similar performance, with both methods having a better RPD in ten of cases. The average RPD of the IGA over all instances is 6.50%, compared to an average RPD of 10.15% for the HSS. Both algorithms have more difficulties to solve instances where the number of jobs is small compared to the number of machines, with an average RPD of 13.2% for IGA, and 19.9% for HSS. For the other instances the solutions are with an RPD of 2.0% and 3.6% close to optimal.

Table II
COMPARISON OF OUR ITERATED GREEDY ALGORITHM TO HYBRID SCATTER SEARCH OF NASIRI AND KIANFAR [14].

			HSS		IGA	
Name	n	m	APRD	Time	APRD	Time
nas01	15	15	25.15	59.50	**13.76**	25
nas02	15	15	25.08	57.70	**8.88**	25
nas03	15	15	**17.26**	57.80	17.37	25
nas04	15	15	27.80	59.00	**12.90**	25
nas05	15	15	27.86	58.70	**9.20**	25
nas06	15	15	28.26	59.90	**17.51**	25
nas07	15	15	**17.27**	58.60	18.77	25
nas08	15	15	23.27	59.60	**13.64**	25
nas09	15	15	17.18	60.30	**9.58**	25
nas10	15	15	16.32	59.10	**15.59**	25
nas11	20	15	**2.71**	81.10	4.15	35
nas12	20	15	10.32	82.90	**7.98**	35
nas13	20	15	7.28	80.00	**1.71**	35
nas14	20	15	**1.50**	77.80	5.62	35
nas15	20	15	10.64	81.20	**2.02**	35
nas16	20	15	12.22	79.60	**1.11**	35
nas17	20	15	6.58	85.50	**0.02**	35
nas18	20	15	10.89	76.60	**3.12**	35
nas19	20	15	**1.32**	79.60	6.84	35
nas20	20	15	12.27	79.80	**2.30**	35
nas21	20	20	22.43	170.00	**18.73**	80
nas22	20	20	**9.73**	192.40	13.58	80
nas23	20	20	**12.12**	179.10	15.55	80
nas24	20	20	20.90	185.50	**13.42**	80
nas25	20	20	20.56	180.90	**9.69**	80
nas26	20	20	14.44	188.50	**10.38**	80
nas27	20	20	17.17	193.90	**14.19**	80
nas28	20	20	16.47	183.90	**13.56**	80
nas29	20	20	19.35	193.90	**11.65**	80
nas30	20	20	19.49	187.00	**6.42**	80
nas31	30	15	0.79	239.20	**0.01**	100
nas32	30	15	2.75	257.30	**0.12**	100
nas33	30	15	**0.00**	125.30	0.08	100
nas34	30	15	0.60	227.40	**0.01**	100
nas35	30	15	0.87	224.70	**0.15**	100
nas36	30	15	**0.00**	95.50	0.03	100
nas37	30	15	**0.00**	28.90	0.09	100
nas38	30	15	**0.00**	199.10	0.01	100
nas39	30	15	2.49	256.80	**0.00**	100
nas40	30	15	2.18	225.80	**0.03**	100
nas41	30	20	6.00	281.50	**2.91**	125
nas42	30	20	**0.00**	151.40	3.54	125
nas43	30	20	**1.00**	302.00	1.29	125
nas44	30	20	**0.00**	223.20	1.73	125
nas45	30	20	**0.00**	241.90	4.38	125
nas46	30	20	**0.00**	227.00	2.03	125
nas47	30	20	4.43	300.30	**1.15**	125
nas48	30	20	6.12	284.90	**3.71**	125
nas49	30	20	**0.00**	185.70	4.24	125
nas50	30	20	6.18	301.80	**0.50**	125
Averages			10.15	152.6	6.50	73.0

VII. CONCLUSIONS

We have presented an extension of the shifting bottleneck procedure to the partial job shop scheduling problem and shown how it can be applied with an iterated greedy algorithm to obtain a simple but effective heuristic. In comparison with the hybrid scatter search of Nasiri and Kianfar [14] in obtains mostly better results in a comparable time, which an average RPD of about 4% lower.

REFERENCES

[1] Joseph Adams, Egon Balas, and Daniel Zawack. "The shifting bottleneck procedure for job shop scheduling". In: *Management science* 34.3 (1988), pp. 391–401.

[2] David Applegate and William Cook. "A computational study of the job-shop scheduling problem". In: *ORSA Journal on computing* 3.2 (1991), pp. 149–156.

[3] Egon Balas and Alkis Vazacopoulos. "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling". In: *Management Science* 44.2 (1998), pp. 262–275.

[4] Peter Brucker et al. "A branch & bound algorithm for the open-shop problem". In: *Discrete Applied Mathematics* 76.1 (1997), pp. 43–59.

[5] Peter Brucker, Bernd Jurisch, and Bernd Sievers. "A branch and bound algorithm for the job-shop schedul-

ing problem”. In: *Discrete applied mathematics* 49.1 (1994), pp. 107–127.

[6] Jacques Carlier. “The one-machine sequencing problem”. In: *European Journal of Operational Research* 11.1 (1982), pp. 42 –47.

[7] Jacques Carlier and Eric Pinson. “An algorithm for solving the job-shop problem”. In: *Management science* 35.2 (1989), pp. 164–176.

[8] Bo Chen and Vitaly A Strusevich. “Approximation algorithms for three-machine open shop scheduling”. In: *ORSA Journal on Computing* 5.3 (1993), pp. 321–326.

[9] Henry Fisher and Gerald L Thompson. “Probabilistic learning combinations of local job-shop scheduling rules”. In: *Industrial scheduling* (1963), pp. 225–251.

[10] Teofilo Gonzalez and Sartaj Sahni. “Open shop scheduling to minimize finish time”. In: *Journal of the ACM (JACM)* 23.4 (1976), pp. 665–679.

[11] GF Italiano. “Transitive closure for dynamic graphs”. In: *Proc. 24th Ann. Allerton Conf. on Communication, Control and Computing*. 1986, pp. 29–38.

[12] Teruo Masuda, Hiroaki Ishii, and Toshio Nishida. “The mixed shop scheduling problem”. In: *Discrete Applied Mathematics* 11.2 (1985), pp. 175 –186.

[13] Mohammad Mahdi Nasiri and Farhad Kianfar. “A GA/TS algorithm for the stage shop scheduling problem”. In: *Computers & Industrial Engineering* 61.1 (2011), pp. 161 –170.

[14] MohammadMahdi Nasiri and Farhad Kianfar. “A hybrid scatter search for the partial job shop scheduling problem”. English. In: *The International Journal of Advanced Manufacturing Technology* 52.9-12 (2011), pp. 1031–1038.

[15] Eugeniusz Nowicki and Czeslaw Smutnicki. “A Fast Taboo Search Algorithm for the Job Shop Problem”. In: *Management Science* 42.6 (1996), pp. 797–813.

[16] Eugeniusz Nowicki and Czeslaw Smutnicki. “An Advanced Tabu Search Algorithm for the Job Shop Problem”. In: *J. of Scheduling* 8.2 (Apr. 2005), pp. 145–159.

[17] Sigrid Knust Peter Brucker. *Complexity results for scheduling problems*. 2012. URL: http://www.informatik.uni-osnabrueck.de/knust/class/.

[18] Ferdinando Pezzella and Emanuela Merelli. “A tabu search method guided by shifting bottleneck for the job shop scheduling problem”. In: *European Journal of Operational Research* 120.2 (2000), pp. 297–310.

[19] Christian Prins. “Competitive genetic algorithms for the open-shop scheduling problem”. In: *Mathematical methods of operations research* 52.3 (2000), pp. 389–411.

[20] A. Ramudhin and P. Marier. “The Generalized Shifting Bottleneck Procedure”. In: *European Journal of Operational Research* 93.1 (1996-08-23T00:00:00), pp. 34–48.

[21] Rubén Ruiz and Thomas Stützle. “A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem”. In: *European Journal of Operational Research* 177.3 (2007), pp. 2033–2049.

[22] E. Taillard. “Benchmarks for basic scheduling problems”. In: *European Journal of Operational Research* 64.2 (1993), pp. 278 –285.

[23] Jean-Paul Watson, Adele E Howe, and L Darrell Whitley. “Deconstructing Nowicki and Smutnicki’s i-TSAB tabu search algorithm for the job-shop scheduling problem”. In: *Computers & Operations Research* 33.9 (2006), pp. 2623–2644.