Contents

13. Job shops **1**
Johann L. Hurink, Jan Karel Lenstra, David B. Shmoys
13.1. The disjunctive graph model for $J||C_{\max}$ 1
13.2. Approximation algorithms: computing good solutions 3
13.3. Approximation algorithms: geometric results 15

13

Job shops

Johann L. Hurink
Twente University

Jan Karel Lenstra
Centrum Wiskunde & Informatica

David B. Shmoys
Cornell University

13.1. The disjunctive graph model for $J||C_{\max}$

In the job shop scheduling problem, each job J_j consists of a chain of operations (i.e., an ordered sequence of operations), where each operation is specified to be processed on a particular machine for a specified length of time (without interruption). Each machine can process at most one operation at a time. For $J||C_{\max}$, the objective is to minimize the makespan, that is, the time by which all jobs are completed.

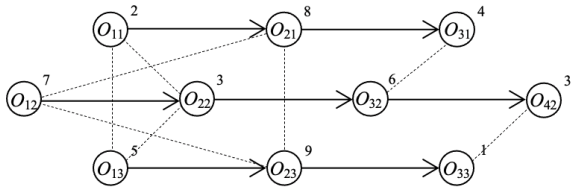
The above description does not reveal much of the structure of this problem type. An illuminating problem representation is provided by the *disjunctive graph model*.

Given an instance of $J||C_{\max}$, the corresponding disjunctive graph is defined as follows. For every operation O_{ij}, there is a vertex, with a weight p_{ij}. For every two consecutive operations of the same job, there is a (directed) arc. For every two operations that require the same machine, there is an (undirected) edge. Thus, the arcs represent the job precedence constraints, and the edges represent the machine capacity constraints.

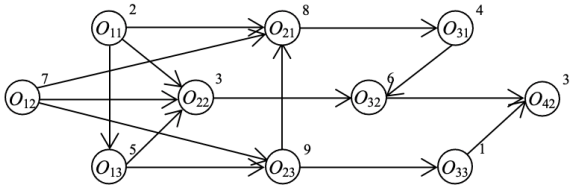
The basic scheduling decision is to impose an ordering on a pair of operations on the same machine. In the disjunctive graph, this corresponds to orienting the edge in question, in one way or the other. A schedule is obtained by orienting all of the edges. The schedule is feasible if the resulting directed graph is acyclic, and its

J_j	m_j	μ_{1j}	μ_{2j}	μ_{3j}	μ_{4j}	p_{1j}	p_{2j}	p_{3j}	p_{4j}
J_1	3	M_1	M_2	M_3	–	2	8	4	–
J_2	4	M_2	M_1	M_3	M_4	7	3	6	3
J_3	3	M_1	M_2	M_4	–	5	9	1	–

(a)



(b)



(c)

Figure 13.1. Job shop scheduling problem. (a) Instance. (b) Instance, represented as a disjunctive graph. (c) Feasible schedule, represented as an acyclic directed graph.

length is obviously equal to the weight of a maximum weight path in this graph.

The job shop scheduling problem has now been formulated as the problem of finding an orientation of the edges of a disjunctive graph that minimizes the maximum path weight. We refer to Figure 13.1 for an example.

13.2. Approximation algorithms: computing good solutions

In this section we deal with the second class of approaches mentioned in Chapter 2 to cope with NP-complete problems: we present heuristic methods to solve the job shop problem. These methods enter the scene if the computation time to get solutions is limited or the instances to be considered get larger. In both cases, branch-and-bound methods may fail. Depending on the available time or the size of the instances, simple constructive heuristics or more time consuming iterative methods may be used. The main focus of this section lies on the second class of heuristics. Starting from an initial solution achieved e.g., by a constructive heuristic, the iterative methods change the given solution slightly and repeat this process iteratively hoping that finally a good solution results. Based on the iterative process (searching) and the restriction to slight changes (local), these methods are called *local search* methods. The local search approaches presented here for the job shop problem work very well in practice, although they give no proven performance guarantee. However, getting approximation methods with a proven performance guarantee for the job shop problem is limited. Since it has been proven that determining whether or not a schedule of length ≤ 4 exists is NP-complete, it follows that the problem of finding an approximation algorithm with a performance guarantee better than $\frac{5}{4}$ is NP-hard.

This section is organized as follows. In the following subsection we will give a brief description of constructive heuristics for the job shop problem. In addition to their value as stand alone methods, these methods can be used to calculate initial solutions for local search procedures, which form the contents of Subsection 13.2.2. Finally, in Subsection 13.2.3 some implementational issues are discussed.

13.2.1. Constructive methods

In this part we concentrate on constructive heuristics for the job shop problem. In addition to simple methods based on priority rules, we also discuss a more elaborate method called *shifting bottleneck* procedure.

As mentioned in Section 14.1, the main scheduling decision is to define processing orders on the machines leading to complete orientations. Given such a complete orientation, by longest path calculations we obtain a corresponding schedule in which an operation starts at the time where either its job or machine predecessor is finished or, if both do not exist, at time 0. Schedules that have this property (no operation can start earlier without changing at least one machine order) are called *semi-active* schedules. Two important subclasses of the semi-active schedules are the

active schedules and the *non-delay* schedules. The first class contains only schedules (complete orientations) where it is not possible to start an operation earlier without delaying the start of another operation. The second class is a subset of the first and contains only schedules where a machine may be idle only if no jobs are available for processing. Whereas the set of active schedules is still a dominating set (i.e., it contains at least one optimal schedule), optimal schedules do not have to be a non-delay schedules (see Exercise 13.1).

Most of the priority driven heuristics for the job shop problem generate active or non-delay schedules. The basic principle of all these methods is the same and can also be found in the method to calculate upper bounds within branch-and-bound procedures. Iteratively repeat the following steps:

1. estimate the set C of operations for which the job predecessors have already been scheduled,
2. select an operation in C and schedule it next on its machine.

Thus, in each iteration one operation is added to the given partial schedule as the last operation on its machine. In principle, Step 2 is always realized as follows:

2.1 determine a subset $\bar{C} \subset C$,

2.2 select an operation from $\bar{C}$ using a priority rule.

Depending on the realization of these two sub-steps, different constructive heuristics result. The priority rules in Step 2.2 may be simple rules like shortest processing time (SPT), longest processing time (LPT), longest remaining processing time (LRPT) etc., or may be more complex rules depending on a lower bound for the makespan assuming that the given operation is scheduled next. The selection of the subset $\bar{C} \subset C$ in Sub-step 2.1 may be realized in three different ways:

- Select $\bar{C}$ as the subset of operations from C that can start earliest (i.e., all operations in $\bar{C}$ can start at the same time if added to the current partial schedule and no other operation from C can start earlier).
  In this case no idle time on machines occurs if operations are available for processing and, thus, the resulting schedule is a non-delay schedule.
- Let u be an operation from C that finishes earliest if added to the current partial schedule and let i be the machine on which u has to be scheduled. Select $\bar{C}$ as the subset of operations from C that have to be scheduled on machine i and can start before u would finish.
  In this case, no operation can be processed earlier without delaying another operation and, thus, the resulting schedule is an active schedule.
- Select the complete set C (i.e., $\bar{C} = C$).
  In this case the resulting schedule will be a semi-active schedule.

From a theoretical point of view, the second choice is preferable since the set of non-delay schedules generated by the first choice in general does not have to contain an

$M := \{M_1, \ldots, M_m\}$; $M_0 := \emptyset$;
REPEAT
    choose a machine M_i from M;
    fix precedence relations for M_i;
    $M := M \setminus \{M_i\}$; $M_0 := M_0 \cup \{M_i\}$;
    re-optimize machines from M_0;
UNTIL $M = \emptyset$.

Figure 13.2. Basic structure of the shifting bottleneck heuristic.

optimal schedule and since the additional solutions that can be generated by the third choice give no further improvements. However, the first and third choice reflect the way planning in production plants is often carried out.

A more elaborate constructive heuristic for the job shop problem is the *shifting bottleneck heuristic* (SBH). In the remainder of this subsection we describe its basic version.

The basic idea of the SBH is to schedule the m machines one by one. In each stage the disjunctions belonging to a specific machine are fixed taking into account the selections already made for scheduled machines. Furthermore, at the end of each stage, the partial schedule belonging to the fixed machines is re-optimized; see Figure 13.2 for the basic structure of SBH.

The choice of the machine to be fixed next, the scheduling of this machine, and the re-optimization all rely on the same process: given a partial schedule via fixed disjunctions for a subset of machines, analyze the situation on a specific machine. The fixed disjunctions lead to earliest possible start times (release dates) and minimal delivery times (which can be transformed into due dates, see Chapter 3) for the operations to be scheduled on the specified machine. The resulting problem is a single-machine head-body-tail problem, for which an efficient branch-and-bound method exists to determine the minimal maximum lateness. The machine to be fixed next is chosen to be the machine with the largest maximum lateness of the head-body-tail problem (i.e., the bottleneck machine). On this machine, the disjunctions are fixed according to the optimal solution of the head-body-tail problem. For the re-optimization, the fixed machines (machines from M_0) are treated consecutively by first removing their fixed disjunctions and then rescheduling them according to the optimal solution of the head-body-tail problem that results after dropping the given schedule on the machine.

The SBH is an effective heuristic for the job shop problem. Several variants of the method and integrations of the method in other heuristics have been presented. The variations mainly differ in the way how the re-optimization is organized. Finally, it is worthwhile to mention that the basic structure of SBH also can be used to solve generalizations of the job shop problem.

choose an initial solution $s \in S$;
REPEAT
    choose a solution $s' \in N(s)$;
    $s := s'$;
UNTIL stopping criteria.

Figure 13.3. Basic structure of local search.

13.2.2. Local search

The methods presented in the previous subsection basically construct one feasible solution. However, as in the re-optimization step of SBH, it may be worthwhile to change this solution somehow and achieve other, hopefully better, feasible solutions. Local search methods form an important and widely used class of such approaches.

In the following we first give a general overview of these methods and, afterwards, show how they can be adopted to the job shop problem.

General overview. Local search methods form a generic class of heuristics. Generally speaking, local search methods iteratively move through the set of feasible solution. Based on the current and maybe previously visited solutions, a new solution is chosen. The choice of the new solution is restricted to solutions that are somehow close to the current solution (in the 'neighborhood' of the current solution).

Looking at the general structure of local search, we may conclude that one of the basic ingredient of local search is the *neighborhood*. The neighborhood determines in an essential way how the search process will behave. To define a neighborhood on a set S of feasible solutions, we have to specify a corresponding set of *neighbors* $N(s) \subset S$ for each solution $s \in S$. In principle, these sets $N(s)$ may be arbitrary subsets of S. However, following the idea of local search, these solutions should be somehow 'similar' to s. A systematic way of defining neighborhoods is to specify a set OP of *operators* $op : S \to S$ which perturb solutions in some way. In this case, the set of neighbors of a solution s is defined by $\mathcal{N}(s) = \{op(s) | op \in OP\}$.

Using the notion of neighborhoods, the basic structure of local search is given in Figure 13.3. This basic local search algorithm may be made concrete in various ways. Depending on the method of choosing solutions from the neighborhood of the current solution and the way in which the stopping criteria are defined, we get different local search methods.

A very intuitive and natural way to make concrete the choice of a solution from the neighborhood of the current solution and the stopping criteria in the algorithm of Figure 13.3 is to make in each step locally the best choice and to stop if this does not lead to an improvement. This method is called *iterative improvement* and has its roots in the year 1958. Its main characteristic is that it stops if a local optimum with respect to the given neighborhood has been reached. Generally, this local optima depends on the chosen initial solution and no information is available on how much

the quality of this solution differs from that of the global optimum.

It took until the mid-1980s before alternative local search methods were developed that did not terminate at the first local optimum. The first method of this type was *simulated annealing*. This method has two stochastic elements. First, a solution is chosen from the set of neighbors of the current solution according to some given distribution (each neighbor usually has the same probability). Afterwards, depending on the difference between the objective values of the chosen and the current solution, it is decided whether we move to the chosen solution or stay with the current solution. If the chosen solution has a better objective value, we always move to this solution. Otherwise, we move to this solution with a probability that depends on the difference between the two objective values. More precisely, if s_1 denotes the current solution and s_2 the chosen solution, we move to s_2 with probability

$$p(s_1, s_2) = e^{\min\{c(s_1)-c(s_2),0\}/c}, \tag{13.1}$$

where $c(s)$ denotes the objective value of a solution s. The parameter c is a positive control parameter that decreases with increasing number of iterations and converges to 0. This has the effect that the probability of moving to a solution with larger objective value decreases in the course of time and that, finally, almost only improving solutions are accepted. Furthermore, the probability (13.1) has the property that large deteriorations of the objective function are accepted with lower probability than small deteriorations. Under certain conditions for the neighborhoods and the way in which the control parameter c is decreased, it is possible to prove that simulated annealing is asymptotically an optimization algorithm. The proof is based on results on Markov chains.

Another local search method that allows nonimproving moves is the *tabu search* method. Its central idea is to deterministically guide the local search and to use information from the past to avoid cycling. The most straightforward realization of this idea is to always choose the best solution from the neighborhood (steepest descent or, if no descent is possible, mildest ascent), but to reduce the neighborhood by all solutions already visited. However, for practical reasons it is not possible to keep track of all solutions already visited and, thus, the process of reducing the set of neighbors is realized in a different way. Generally, the search for the next solution $s' \in N(s)$ is restricted to a subset $N_{red} \subset N(s)$, whereby either a set of operators from OP or solutions with certain properties are forbidden (tabu). These restrictions vary in time and are chosen so that it is not possible to revisit a certain solution within a given number of iterations. More precisely, depending on the last k moves, a set of operators or a set of properties is defined as tabu. Since by this approach not only already visited solutions but also other solutions may be excluded from the search process, so-called *aspiration criteria* are used to overrule the tabu status of a solution. A simple aspiration criterion is that a solution has a better objective value than the best solution found thus far (such a solution has not been visited before and, thus, should not be tabu). However, more elaborate aspiration criteria can also be used.

Another type of solution approaches to solve combinatorial optimization problems is given by *genetic algorithms*. These methods are based on principles from

population genetics and the theory of evolution. Roughly speaking, genetic algorithms start with some subset of feasible solutions (a population) and iteratively replace the current population by a next population. Thus, in principle, genetic algorithms are local search methods on subsets of solutions.

The process of building new populations is organized by two main operators. The first is called *mutation* and is applied with some given probability to each solution of the current population. A mutation changes the given solution slightly. Mostly, there are several alternatives in changing the solution and the choice for one of them is done randomly. Thus, a mutation is nothing but a random selection of one solution in the neighborhood of the given solution, whereby the neighborhood is defined by the possible changes mutations can perform. The second operator is called *recombination*. Recombination is used to produce a subset of new solutions (offsprings). To achieve this goal, pairs of solutions (parents) are selected from the current population and for each pair two new solutions (children) are constructed by 'shuffling' the given solutions. Again, there are several ways of shuffling solutions and the choice between these alternatives will be done randomly. For selecting pairs of solutions the quality of the solutions (objective values) plays an important role. Often, the parents are selected randomly using a distribution on the current population that depends on the quality of the solutions.

After applying mutations and recombinations a new population has to be determined. This new population will be a subset of the union of the population after mutation and the set of new solutions resulting from recombination. It is often chosen as the subset of the k best solutions, where k denotes the cardinality of the population.

Application to the job shop problem. One main step for applying local search to a job shop problem is to decide on the underlying neighborhood. Most of the commonly used neighborhoods rely on the representation of solutions by complete orientations and change the orientation of one or more edges. In this context, two properties are helpful:

- reversing the orientation of an edge on a *critical path* of a feasible solution leads again to a feasible solution (see Exercise 13.2),
- only changes that affect the first or last operation of a *block* have the potential to lead to an improving solution.

A critical path is a longest path in the directed graph corresponding to the feasible solution, and a block is a maximal sequence of adjacent operations that are processed on the same machine and belong to a critical path. The first property gives a sufficient condition to get feasible neighbors and the second indicates which solutions are non-promising with respect to the objective value. However, due to the nature of local search even a bad solution with respect to the objective value may be a good solution for the further search process. Furthermore, the given sufficient condition on feasibility may be much too restrictive for the search. Therefore, only a few of the following neighborhoods rely completely on these properties.

- the critical path interchange neighborhood N_{inter}^{CP}
  N_{inter}^{CP} allows the interchange of two adjacent operations of a block on the critical path (i.e., the reversion of the corresponding edge) and, thus, is based on the first property.

- the end-of-block interchange neighborhood $N_{inter}^{end-block}$
  $N_{inter}^{end-block}$ is a sub-neighborhood of N_{inter}^{CP}, where only the interchange of the first two operations or the last two operations of a block are allowed. This reduction is based on the second property, since the interchange of two internal adjacent operations of a block cannot lead to an improved solution.

- the critical path permutation of three neighborhood N_{3-perm}^{CP}
  N_{3-perm}^{CP} is an extension of N_{inter}^{CP} and considers, in addition to the interchange of two adjacent operations of a block, the effect if after this interchange the machine predecessor or machine successor is interchanged with the two operations. More precisely, for four consecutive operations p,v,w,s on machine with v,w lying on a critical path, the following sequences are considered as neighbors if they lead to a feasible solution: (p,w,v,s), (w,p,v,s), (w,v,p,s), (p,w,s,v), and (p,s,w,v).

- the end-of-block permutation of three neighborhood $N_{3-perm}^{end-block}$
  Again, $N_{3-perm}^{end-block}$ is the sub-neighborhood of N_{3-perm}^{CP}, where only neighboring operations v,w are considered which are the first two operations or the last two operations of a block.

- the shift to the end of a block neighborhood N_{shift}^{block}
  N_{shift}^{block} allows a shift of an operation of a block immediately in front of or after the other operations of its block, provided that the resulting solution is feasible. Otherwise, the operation is moved as far as possible to the beginning or end of the block resulting still in a feasible solution. Again, this neighborhood is based on the second property.

- the shift on the same machine neighborhood $N_{shift}^{machine}$
  $N_{shift}^{machine}$ executes shifts for two operations v,w occurring on a critical path and scheduled on the same machine. If v is scheduled before w on the critical path, shifting v directly after w and shifting w directly before v are possible neighbors. However, if both the machine predecessor of v and the machine successor of w are on the critical path, the resulting neighbor cannot improve the current solution (see Exercise 13.3) and is not included in the neighborhood.

- the end-of-block three interchange neighborhood $N_{3-inter}^{end-block}$
  The basic idea of this neighborhood is the same as for $N_{inter}^{end-block}$. However, under certain conditions, not only the two adjacent operations v,w are interchanged but also the job successor of the first operation (v) is interchanged

with its machine successor and one of the job predecessors of the second operation (w) is interchanged with its machine predecessor.

- the shifting bottleneck neighborhood N_{SBH}
  A neighbored solution for N_{SBH} is obtained by removing all orientations on a machine with at least one operation on a critical path and replacing them by any other orientation leading to a feasible solution.

- the several machine shifting bottleneck neighborhood $N_{SBH}^{several}$
  In contrast to N_{SBH}, not only the orientation on one machine but the orientations on $m-t$ machines are replaced for this neighborhood (t is a small number depending on m).

Most of the presented neighborhoods change the processing order on only one machine. Only $N_{3-inter}^{end-block}$ and $N_{SBH}^{several}$ change orders on more than one machine.

For local search methods like iterative improvement, simulated annealing, and tabu search a choice for one of the above defined neighborhoods is sufficient to describe the search process. However, for genetic algorithms the situation is a bit more complex. Whereas the above neighborhoods may be used for mutations, also operators for the recombination have to be given. One possible realization of the recombination, which is somehow similar to the above defined neighborhoods is given as follows:

- recombination by interchanges R_{inter}^{random}
  Given two solutions S_1 and S_2 an offspring is constructed by repeating $\lceil nm/2 \rceil$ times: select randomly an edge $\{v,w\}$; if this edge is oriented in S_1 and S_2 in different ways, and if it belongs in S_1 to a critical path, interchange v and w in S_1. The resulting solution S_1 is the offspring.

Other realizations of the recombination rely on constructive heuristics for the job shop problem. The simplest one uses the priority driven heuristic presented in Subsection 13.2.1, which led to an active schedule:

- recombination by constructing active schedules R_{heur}^{active}
  Given two solutions S_1 and S_2 an offspring is constructed by a priority driven constructive heuristic using the following priority rule for the operations in $\bar{C}$ ($\bar{C}$ is determined by the second possibility given in Subsection 13.2.1): choose the operation which starts first in S_1 with probability $(1-\varepsilon)/2$, the operation which starts first in S_2 with probability $(1-\varepsilon)/2$, and randomly one of the other operations with probability ε (ε is a given small positive value).

All the presented neighborhoods and recombination methods may be realized using the mixed graph representation of the job shop problem. However, there also exists local search methods, especially genetic algorithms, which use different representations. One of these representations based on a constructive heuristic and perturbation of the data will be given in the notes.

Combining one of the sketched versions of local search with one of the above defined neighborhoods leads to a suitable heuristic for solving the job shop problem. However, to get efficient methods often some variations or adaptions of the methods are useful. In the following, some variations or adaptions of local search methods proposed for the job shop problem are given.

- Iterative improvement:

  This method stops in the first local optima and, therefore, evaluates in general only a few solutions. Thus, to make it somehow compatible to the other local search methods, a multi-start strategy (i.e., start the search for several different starting solutions) has to be applied.

- Simulated annealing:

  Besides the 'pure' variant also backtracking strategies (after a certain number of iterations without improving the best found solution so far, move back to the best found solution and restart the search) or a combination with iterative improvement (do not evaluate a neighbor directly, but investigate its potential for the further search by applying (a few) iterations of iterative improvement to the neighbor before evaluating it) have been proposed.

- Tabu search:

  For tabu search several variations have been proposed in the literature. The first type of variation depends on the definition of the tabu status. In general a list (tabu list) keeps track of the last k moves and the entries of the tabu list are used to define operators or solutions as tabu. On the one hand, the length of this list (k) may be fixed or may vary over time. On the other hand, the choice which operators/solutions are declared as tabu in dependence of the used operator may vary and also the strictness of the tabu status may be different (e.g. aspiration criteria).

  Another variation of tabu search results from the incorporation of backtracking strategies. Here one can vary the conditions for a backtracking move (e.g. maximum number of moves without improving the best found solution), the choice for the solution where the search is restarted (e.g. the best solution), and the restart conditions (e.g. use the next best neighbor or use the same solution but with a different length of the tabu list).

  Finally, within tabu search the search for the best non-tabu neighbor may be speeded up by replacing the exact evaluation of the objective value of neighbored solutions by quickly computed bounds on the change of the objective value.

- Guided local search:

  Guided local search uses elements from tabu search and back tracking and was first applied to the job shop problem by Balas and Vazacopoulos. In each step,

a neighborhood tree of solutions is constructed, where the current solution serves as root. For each node of the tree, a set of descendants is created by generating neighbored solutions using some neighborhood structure. After creating the tree, one of its best solutions is chosen as the new current solution and, thus, serves as the root of the next tree.

The main ingredient of guided local search is the construction of the neighborhood tree. Here the depth of the tree, the number of descendants per node, and the choice of the descendants play an important role. To bound the computation times, the depth and the number of descendants are kept small and the selection of descendants is based on lower bounds on the completion time of the neighboring solution and not on an exact evaluation of the makespan. Furthermore, within the generation process it is guaranteed that on a path to a node from the root, none of the neighborhood changes made is reversed.

- Genetic algorithms:

  Several proposed genetic algorithms for the job shop problem incorporate other local search methods. In the simplest variants, the solutions generated by mutations and recombinations are locally optimized by iterative improvement or improved by fast versions of simulated annealing.

  Other genetic algorithms used priority sequences for the machines as solution representation. For such a representation, a corresponding solution is achieved by applying some constructive heuristic based on priority rules. Somehow, these representations are similar to complete orientations. However, the main difference is that the set of sequences may not lead to a complete orientation since the resulting mixed graph may contain cycles. The advantage of the priority sequence based representation is that within the mutation and recombination process one does not have to deal with this feasibility question, since the application of the constructive heuristic acts as a repair mechanism. Some authors suggest that, after the application of the constructive heuristic, to replace the given priority sequence with the concrete sequences on the machines in the constructed schedule.

13.2.3. Implementation

Many constructive heuristics for the job-shop problem rely on priority-based dispatching rules. However, the quality of the solutions achieved by these methods is not very good and these methods are mostly just used within other methods, e.g., to calculate initial solutions.

The more successful constructive method, the shifting bottleneck heuristic, was proposed by Adams et al. In the original version, in each iteration they re-optimize all machines from M_0 three times in three cycles. In the first cycle the machines are rescheduled in the order they were inserted in M_0 and in last two cycles the machines are rescheduled in the order of their maximum lateness in the previous

cycle. Note, that these rescheduling operations of the machines can be seen as one neighborhood step in N_{SBH}, where the current schedule on the machine is replaced by its best neighbor. Applegate & Cook [1991] present a variation of this method by not fixing the number of cycles for the re-optimization but repeating this step until for no machine an improvement is obtained.

For the job shop problem a large number of different local search methods has been proposed in the literature. It starts with more or less basic versions of the different search methods using a single neighborhood.

- Aarts et al. test iterative improvement with neighborhoods N_{inter}^{CP} and $N_{3-inter}^{end-block}$.
- The first versions of simulated annealing are given by Matsuo et al. and Van Laarhoven et al. Matsuo et al. use neighborhood $N_{3-inter}^{end-block}$ but incorporate also some features of iterative improvement. Van Laarhoven et al. use neighborhoods N_{inter}^{CP} and $N_{3-inter}^{end-block}$.
- Tabu search was first applied by Dell'Amico & Trubian and Taillard. Dell'Amico & Trubian use neighborhoods $N_{3-perm}^{end-block}$ and N_{shift}^{block} and a tabu list of variable length, whereas Taillard uses neighborhood N_{inter}^{CP} and tabu list of fixed length. Furthermore, Taillard does not estimate the makespan of all neighbored solutions exactly but uses lower bounds.
- Balas & Vazacopoulos proposed guided local search using the neighborhood $N_{shift}^{machine}$. They bound the size of the tree by limiting the number of descendants, by fixing part of the orientations, and by bounding the depth of the tree by a logarithmic function of the number of operations.
- The first genetic algorithms for the job shop problem were proposed by Davis and Falkenauer & Bouffouix. Both use priority sequences to represent solutions and a heuristic to produce a non-delay schedule from a given representations.

Besides these first, and often basic, local search approaches, various other, more elaborate, applications of local search to the job shop problem are given in the literature. Several of these approaches combine different local search methods or combine local search with the shifting bottleneck heuristic. For genetic algorithms besides the mentioned representation based on orientations or priority sequences, also other, less straightforward, representations have been proposed. E.g., Dorndorf & Pesch use a sequence of priority rules as representation. Using this sequence a schedule is built up by a simple priority driven heuristic, where in case of a conflict in the ith iteration the priority rule on position i of the sequence is used to solve the conflict. Ponnambalam et al. present a comparative evaluation of different representations for genetic algorithms for the job shop problem. They compare operation-based, job-based, priority list-based, and priority rule-based representations and come to the conclusion that priority list-based representations lead to the best quality of the solutions and that operation-based representation lead to the smallest computation times.

Empirical evidence supports the general conclusion that hybrid approaches (combination of different methods) are mostly superior to pure approaches. In the remaining of this section we shortly describe the basic ideas behind the three most successful local search approaches for the job shop problem.

Balas & Vazacopoulos combine the shifting bottleneck heuristic with their guided local search approach. In the basic version they realize the re-optimization step of the shifting bottleneck heuristic by their guided local search approach using the current partial schedule as initial solution. The best schedule obtained in this processed is used to continue with the shifting bottleneck heuristic. Using the solution of this basic version as initial solutions two other variants are proposed. In the first, called *iterated guided local search*, iterations of re-optimization cycles are carried out until no improvement is found. Each cycle removes the orientation of one machine, applies guided local search for a limited number of trees, adds the removed machine again, and applies guided local search to the complete schedule for a limited number of trees. The second variant, called *reiterated guided local search*, uses the solution of the first variant as an initial solution and repeats a fixed number of cycles, where the orientation of $\sqrt{m}$ machines (randomly chosen) are removed, guided local search with a limited number of trees is applied to the resulting partial schedule, and the removed machines are added back by applying the basic version of the method.

Pezzella & Merelli also use elements from the shifting bottleneck heuristic and combine them with tabu search elements. The initial solution is achieved by a standard shifting bottleneck heuristic. Afterwards tabu search is applied using as neighborhood the neighborhood N^{CP}_{inter} extended by some further neighbors achieved via shifts within blocks. Within the tabu search approach again an element of the shifting bottleneck heuristic is used: every time a new best solution has been found, a re-optimization as within the shifting bottleneck heuristic is applied to all machines involved in the critical path.

The third, and at the moment the best, approach is given by Nowicki & Smutnicki. Their approach is a tabu search approach using backtracking and the underlying neighborhood is $N^{send-block}_{inter}$. Each time a certain number of iterations yield no improvement to the best solution found, the method restarts from the best solution found with a different neighborhood move. Furthermore, if from one solution all possible restarts have been carried out, this best solution is no longer used for restarts, but the former best solution will act as base for the restarts.

Exercises

13.1. Give an instance of the job shop problem, where no optimal schedule is a non-delay schedule.

13.2. Show that reversing the orientation of an edge on a critical path of a feasible solution leads again to a feasible solution.

13.3. For the neighborhood $N^{machine}_{shift}$ solutions resulting from shifts for operations v, w, where both the machine predecessor of v and the machine successor of w are on the critical path cannot improve the given solution.

13.3. Approximation algorithms: geometric results

As was true for both open shops and flow shops, it is possible to derive a surprisingly strong result for the job shop problem using the vector sum theorem, which was proved in Chapter 11. The bound on the absolute error of the approximation algorithm will not be as good as it was for the flow shop problem, but it retains the crucial property: the absolute error is independent of the number of jobs, n, and is bounded by a function of the number of machines, m, the maximum number of operations of a job, $\mu_{\max} = \max_j \mu(j)$, and the maximum processing time of an operation, $p_{\max} = \max_{h,j} p_{hj}$. More precisely, we will prove the following result.

Theorem 13.1. *For any instance of $J||C_{\max}$, a schedule of length at most $C^*_{\max} + O(m\mu^3_{\max} p_{\max})$ can be found in polynomial time.*

This theorem is significantly more difficult to prove than Theorem 13.3, the analogous result for flow shop scheduling. In fact, we will only prove that it holds under certain assumptions about the input. We assume that each job has the same number of operations and that each machine has the same load; that is, $\mu(j) = \mu$ for each J_j and, if $\Pi_i = \sum_{\iota(h,j)=i} p_{hj}$ and $\Pi_{\max} = \max_i \Pi_i$, then $\Pi_i = \Pi_{\max}$ for each M_i. The algorithm will deliver a schedule of length $\Pi_{\max} + O(m\mu^3 p_{\max})$. Any instance can be modified to one satisfying these assumptions, without increasing the bound guaranteed; but this extension is left to the reader (see Exercise 13.4).

Since the proof is rather intricate, we will describe the main ideas before proceeding to the details. Focus on the set of the hth operations of jobs that must be performed on machine M_i, and let Π_{hi} denote their total processing time. The key step in the algorithm will be to order the jobs in such a way that, for any r consecutive jobs, the total processing time of their operations that belong to this set is approximately equal to $(r/n)\Pi_{hi}$. Note that a job need not have its hth operation on M_i, in which case it contributes 0 to the total. For simplicity of notation, assume that the jobs are indexed in this order.

We will use the ordering to partition the job set into n/r sets of r consecutive jobs each. The value of r will be chosen so as to ensure that the proposed schedule is feasible on the one hand and sufficiently short on the other hand; for the time being, simply assume that n/r is integral. Call the job set $\{J_{(g-1)r+1}, ..., J_{gr}\}$ *group* g, for $g = 1, ..., n/r$.

The schedule for M_i will consist of a sequence of $n/r + \mu - 1$ *blocks* (cf. Figure 13.4). In the first block, we schedule the first operations of the jobs in group 1 that must go on M_i; in the second block, there are first operations from group 2 and second operations from group 1; and so on. A typical block b contains first operations from group b, second operations from group $b-1$, up to μth operations from group $b-\mu+1$. The last group does not start its first operations until block n/r, and thus finishes only in block $n/r+\mu-1$.

Note that, for any b $(b = 1, ..., n/r+\mu-1)$ and j $(j = 1, ..., n)$, there is at most one machine that processes an operation of J_j in its bth block. Now consider the total processing time in block b on M_i, with $\mu \leq b \leq n/r$. The ordering ensures that the hth

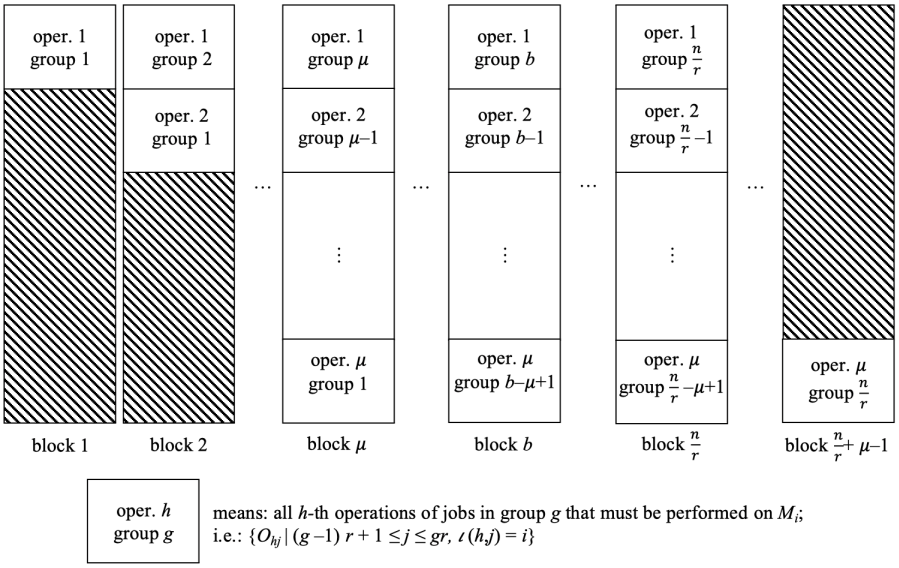


Figure 13.4. Schedule for M_i.

operations from group $b-h+1$ that go on M_i total roughly $(r/n)\Pi_{hi}$, for $h=1,...,\mu$. These sum, even more roughly, to $(r/n)\sum_{h=1}^{\mu}\Pi_{hi}=(r/n)\Pi_i=(r/n)\Pi_{\max}$. However, this calculation does not apply to, for example, the second block, which contains only first and second operations, summing roughly to $(r/n)(\Pi_{1i}+\Pi_{2i})$. We correct this by adding idle time to certain blocks: for each block without hth operations, we introduce idle time of total length $(r/n)\Pi_{hi}$. For example, in the second block we introduce idle time totaling $(r/n)\sum_{h=3}^{\mu}\Pi_{hi}$. As a result, the length of each block on each machine is roughly equal to $(r/n)\Pi_{\max}$.

An intuitive interpretation of this schedule is that we try to group the jobs so that their operations can be synchronized and pipelined. The synchronization is achieved because each block takes roughly the same amount of time on each machine, and the pipelining is achieved because the starting times of the jobs are staggered by groups. What is the length of this schedule? Since each block is roughly of length $(r/n)\Pi_{\max}$, and there are $n/r+\mu-1$ blocks, we get a schedule of length

$$\Pi_{\max}+\frac{r}{n}(\mu-1)\Pi_{\max}. \tag{13.2}$$

Unfortunately, there is a difference between blocks of roughly equal length, and blocks of equal length. The main problem with roughly equal length blocks is that it is possible that two consecutive operations of the same job are done simultaneously or even out of order. For example, block b on M_i may end somewhat later than

block $b+1$ starts on $M_{i'}$. If O_{hj} is scheduled late within block b on M_i and $O_{h+1,j}$ is scheduled early within block $b+1$ on $M_{i'}$, then $O_{h+1,j}$ may start before O_{hj} is completed. This can be fixed by paying attention to the order in which the operations are performed within a block. We will subdivide each block into r *phases*, and guarantee that all of the operations of the same job are scheduled within the same phase of subsequent blocks. As long as r is large enough, two operations of a job will not overlap. In fact, we will choose r to be roughly equal to $nm\mu^2 p_{\max}/\Pi_{\max}$. If this is substituted into expression (13.2) for the schedule length, we get the claimed result.

After this motivation, giving the proof is just a matter of writing out the precise equations and bounds that correspond to the intuition. We first make explicit the properties of the desired ordering of the jobs. The following notation for the processing times will be convenient:

$$p^i_{hj} = \begin{cases} p_{hj} & \text{if } \iota(h,j) = i, \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 13.2. *For any instance of $J||C_{\max}$, a permutation π of $\{1,...,n\}$ such that*

$$\frac{k}{n}\Pi_{hi} - m\mu p_{\max} \leq \sum_{j=1}^{k} p^i_{h\pi(j)} \leq \frac{k}{n}\Pi_{hi} + m\mu p_{\max}, h = 1,...,\mu, i = 1,...,m, k = 1,...,n,$$

can be found in polynomial time.

Proof. This is a straightforward corollary of Theorem 11.4. For $j = 1,...,n$, let v_j be the $(m\mu)$-dimensional vector with components $p^i_{hj} - (\Pi_{hi}/n)$ ($h = 1,...,\mu$, $i = 1,...,m$). By definition, $\sum_{j=1}^{n} p^i_{hj} = \Pi_{hi}$ for all h and i, so that $\sum_{j=1}^{n} v_j = 0$. Since $||v_j|| \leq p_{\max}$, Theorem 11.4 implies that a permutation π can be found in polynomial time such that

$$-m\mu p_{\max} \leq \sum_{j=1}^{k} p^i_{h\pi(j)} - \frac{k}{n}\Pi_{hi} \leq m\mu p_{\max}, h = 1,...,\mu, i = 1,...,m, k = 1,...,n.$$

This is equivalent to the statement of the lemma. □

We will use this permutation π to construct the schedule. For simplicity of notation, we reindex the jobs so that the identity permutation satisfies the property stated in the lemma. Focus on block b on M_i. Recall that this block consists of the hth operations from group $b-h+1$ that go on M_i, where h ranges over all or part of the set $\{1,...,\mu\}$ (cf. Figure 13.4). Recall also that group $b-h+1$ consists of the jobs $J_{(b-h)r+1}, J_{(b-h)r+2}, ..., J_{(b-h+1)r}$. Hence, the operations processed in block b on M_i

are contained in the following matrix:

$$\begin{bmatrix} O_{1,(b-1)r+1} & O_{1,(b-1)r+2} & \cdots & O_{1,br} \\ O_{2,(b-2)r+1} & O_{2,(b-2)r+2} & \cdots & O_{2,(b-1)r} \\ \vdots & \vdots & & \vdots \\ O_{\mu,(b-\mu)r+1} & O_{\mu,(b-\mu)r+2} & \cdots & O_{\mu,(b-\mu+1)r} \end{bmatrix} \tag{13.3}$$

To make things precise, we note that, for a given block index $b(1 \leq b \leq n/r + \mu - 1)$, the operation index h ranges from $\max\{1, b - n/r + 1\}$ to $\min\{\mu, b\}$. An illegitimate value of h implies a group index g outside the range $\{1, ..., n/r\}$ and a job index outside the range $\{1, ..., n\}$.

Phase s of block b will contain the operations in column s of matrix (13.3), for $s = 1, ..., r$. The schedule for this phase on M_i is formed by considering the operations in column s in order of increasing row index h. If such an operation O_{hj} is not on M_i or, in other words, if $p^i_{hj} = 0$, then continue to the next operation in the column. If an operation O_{hj} is undefined or, in other words, if h is outside its range, then let M_i be idle for Π_{hi}/n units of time. Otherwise, schedule O_{hj} on M_i.

This completes the description of the schedule. We now have to show that it is a feasible schedule, and that its length is within the claimed bound. The following lemma is the key to both propositions. Let z denote the total number of phases of the schedule, i.e., $z = r(n/r + \mu - 1) = n + r(\mu - 1)$, and let C_{it} denote the time at which the first t phases of the schedule are completed on M_i.

Lemma 13.3. *The schedule described above satisfies*

$$\frac{t}{n}\Pi_{\max} - m\mu^2 p_{\max} \leq C_{it} \leq \frac{t}{n}\Pi_{\max} + m\mu^2 p_{\max}, i = 1, ..., m, t = 1, ..., z.$$

Proof. Focus on a machine M_i and a phase t. To obtain these bounds on C_{it}, we first fix h $(1 \leq h \leq \mu)$ and compute the total time allocated to M_i during the first t phases, either to process hth operations or to be idle whenever h is outside the range induced by the phase value. Let us extend the notation p^i_{hj} to include these idle periods:

$$p^i_{hj} = \begin{cases} p^i_{hj} & \text{if } O_{hj} \text{ is defined,} \\ \Pi_{hi}/n & \text{otherwise.} \end{cases}$$

It is easily seen from the above matrix that the (possibly undefined) h th operation that is slated for phase 1 is $O_{h,(1-h)r+1}$. Thus, the total time (both processing and idle) associated with hth operations on M_i during the first t phases is given by $\sum_{j=(1-h)r+1}^{(1-h)r+t} p^i_{hj}$. Suppose that t' of these phases correspond to undefined operations, and that the remaining $t - t'$ correspond to legitimate operations (though not necessarily on M_i). We consider their contributions to this sum separately. Each of the former contributes an idle period of length Π_{hi}/n, which sum to $t'\Pi_{hi}/n$. The latter phases correspond to the first $t - t'$ jobs, and Lemma 13.2 implies that these sum to

within $m\mu p_{\max}$ of $(t-t')\Pi_{hi}/n$. We see that

$$\frac{t}{n}\Pi_{hi} - m\mu p_{\max} \leq \sum_{j=(1-h)r+1}^{(1-h)r+t} p^i_{hj} \leq \frac{t}{n}\Pi_{hi} + m\mu p_{\max}.$$

Summing these inequalities over all possible h, we get

$$C_{it} = \sum_{h=1}^{\mu} \sum_{j=(1-h)r+1}^{(1-h)r+t} p^i_{hj} \geq \sum_{h=1}^{\mu} (\frac{t}{n}\Pi_{hi} - m\mu p_{\max}) = \frac{t}{n}\Pi_{\max} - m\mu^2 p_{\max}$$

and

$$C_{it} = \sum_{h=1}^{\mu} \sum_{j=(1-h)r+1}^{(1-h)r+t} p^i_{hj} \leq \sum_{h=1}^{\mu} (\frac{t}{n}\Pi_{hi} + m\mu p_{\max}) = \frac{t}{n}\Pi_{\max} + m\mu^2 p_{\max}.$$

□

In order to ensure feasibility of the schedule, we choose r so large that, irrespective of the machine, phase $br+s$ finishes no later than phase $(b+1)r+s$ starts. In other words, we want to guarantee that $C_{it} \leq C_{i',t+r-1}$ for any pair $(M_i, M_{i'})$ and any t ($t = 1, ..., z-r$). By Lemma 13.3, it suffices to make sure that

$$\frac{t}{n}\Pi_{\max} + m\mu^2 p_{\max} \leq \frac{t+r-1}{n}\Pi_{\max} - m\mu^2 p_{\max}$$

or, equivalently,

$$2m\mu^2 p_{\max} \leq \frac{r-1}{n}\Pi_{\max}.$$

Therefore, if we set $r = lc2nm\mu^2 p_{\max}/\Pi_{\max} rc + 1$, then we get a feasible schedule.

As for the length of this schedule, we substitute this value of r into the upper bound on C_{iz} given by Lemma 13.3, recalling that $z = n + r(\mu - 1)$. We conclude that the schedule is no longer than

$$\begin{aligned}
&\frac{n+r(\mu-1)}{n}\Pi_{\max} + m\mu^2 p_{\max} \\
&\qquad = \Pi_{\max} + (lc\frac{2nm\mu^2 p_{\max}\Pi_{\max}}{r}c + 1)(\mu-1)\frac{\Pi_{\max} n}{+} m\mu^2 p_{\max} \\
&\qquad = \Pi_{\max} + O(m\mu^3 p_{\max}).
\end{aligned}$$

This completes the proof of Theorem 13.1.

Exercises

13.4. Show that Theorem 13.1 holds for any job shop instance. In particular, show

how to pad the input so that each job has the same number of operations and each machine has the same load, without increasing the bound stated by the theorem. (*Hint*: To balance the load, consider increasing some of the p^i_{hj} for which $\iota(h,j) \neq i$.) Does the proof really require that n/r be integral?

13.5. Consider the special case of the job shop problem when both m and $\mu_{\max}$ are fixed. Show that, for any $\varepsilon > 0$, there exists a $(2+\varepsilon)$-approximation algorithm.

13.6. Consider the following generalization of the job shop problem: each job consists of a set of operations, whose order is constrained by a precedence relation, and yet no two of its operations may be processed simultaneously; the objective is to minimize the maximum completion time. (When the precedence relation of each job is a chain, we have the job shop problem.) Give a polynomial-time algorithm for this problem that delivers a solution of length $C^*_{\max} + O(m\mu^3_{\max} p_{\max})$.

13.7. (a) Consider the special case $J|p_{ij} = 1|C_{\max}$. If one relaxes the capacity constraint of each machine, then there is a schedule of length $\mu_{\max}$, in which each O_{ij} starts at time $i-1$. Now, consider the schedules formed by delaying the start of each J_j by some amount t_j, but then scheduling the operations without idle time; that is, O_{ij} is started at time $t_j + i - 1$. Prove that, if each t_j is chosen independently and uniformly at random in the range from 1 to $\Pi_{\max}$, then with high probability, no more than $4\log(n\mu_{\max})$ operations are assigned to a machine at any time.

(b) Use this to give a randomized algorithm which, for every input, produces a schedule of length $O(\log(n\mu_{\max})C^*_{\max})$ and is expected to run in polynomial time.

(c) By using Theorem 13.1, improve the bound in (b) to $O(\log(m\mu_{\max})C^*_{\max})$.

Notes

13.1. *The disjunctive graph model for* $J||C_{\max}$ was proposed by Roy and Sussmann (1964).

13.2. *Approximation algorithms: computing good solutions.* Haupt (1989) gives a survey of priority-driven heuristics for the job shop problem. Osman and Laporte (1996) and Aarts and Lenstra (1997) survey the literature on local search. Vaessens, Aarts, and Lenstra (1996), Anderson and Potts (1997) and Jain and Meeran (1999) give overviews of the application of local search to the job shop problem, including computational comparisons between approaches.

Adams, Balas, and Zawack (1988) developed the shifting bottleneck heuristic. Applegate and Cook (1991) proposed the variant that does not limit the number of reoptimization cycles.

For specific implementations of local search for job shop scheduling, see Aarts, Van Laarhoven, Lenstra, and Ulder (1994), Matsuo, Suh, and Sullivan (1988), Van Laarhoven, Aarts, and Lenstra (1992), Dell'Amico and Trubian (1993), Taillard (1994), Davis (1985), Falkenauer and Bouffouix (1991), Dorndorf and Pesch (1995), and Ponnambalam, Aravindan, and Sreenivasa Rao (2001). Among the leading contenders are guided local search with shifting bottleneck (Balas and Vazacopoulos, 1998), tabu search guided by shifting bottleneck (Pezzella and Merelli, 2000), and tabu search with backtracking (Nowicki and Smutnicki, 1996).

13.3 *Approximation algorithms: geometric results.* Theorem 13.1 is due to Sevastyanov (1986).