ADVANCED JOB SHOP SCHEDULING METHODS

BUCHMEISTER, B. & PALCIC, I.

Abstract: *Scheduling is a decision-making process used on a regular basis in many manufacturing and service industries. It plays an important role in shop floor planning. Job shop is one of the most popular generalized production systems. A schedule shows the planned time when the processing of a specific job will start and will be completed on each machine that the job requires. Schedule is a timetable for both jobs and machines. Complex and mathematically involved scheduling methods require substantial and extensive knowledge. Since job shop scheduling problems fall mostly into the class of NP-hard problems, they are among the most difficult to formulate and solve. In the chapter, a selection of advanced job shop scheduling methods is represented: filtered beam search, constraint-guided heuristic search and genetic algorithms, all demonstrated with simple examples.*

Key words: *job shop, scheduling, filtered beam search, constraint-guided heuristic search, genetic algorithms*

Authors´ data: Full. Prof. Dr. Sc. **Buchmeister**, B[orut]; Assoc. Prof. Dr. Sc. **Palcic**, I[ztok], University of Maribor, Faculty of Mechanical Engineering, Production Engineering Institute, Smetanova 17, 2000 Maribor, Slovenia, European Union, borut.buchmeister@um.si, iztok.palcic@um.si

This Publication has to be referred as: Buchmeister, B[orut] & Palcic, I[ztok] (2015). Advanced Job Shop Scheduling Methods, Chapter 01 in DAAAM International Scientific Book 2015, pp.001-016, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-05-1, ISSN 1726-9687, Vienna, Austria
DOI: 10.2507/daaam.scibook.2015.01

1. Introduction

On the market, which is shaped by increasingly more demanding customers, the rising number of providers and the competitiveness between them, and right business and production strategy are the deciding factors of a company's success (Baesler et al., 2015). It is not enough only to keep up with the others. Today, success represents sustainability in introducing new standards and orientation toward the customer, to the quality and price of products, to flexibility, agility and promptness, to economising in resources and protection of the environment (Huang, 2010). There are an increasing number of companies, which produce mostly with a job shop system for a known customer. With competitive cost calculation and appropriate product quality, time is becoming the most important factor of business success. This is especially noticeable in job shop production, where adaptability and shortening of flow times decide on the business success or failure of the company (Buchmeister et al., 2004).

Scheduling is as old as humankind is. It is about time and some optimisation. It is an act of defining priority or arranging activities to meet certain requirements, constraints or objectives. Time is always a major constraint. People schedule their activities so that jobs could be accomplished within the available time. Time to get up, time to work, to play, to sleep … Time is a limiting unrecoverable resource and we must schedule our activities to utilise this limited resource in an optimum manner.
As the industrialized world develops, more and more resources are becoming critical. Machines, workers and facilities are now thought of as resources in production or service activities. Scheduling these leads to increased efficiency, utilization and profitability for the company (Yagmahan & Yenisey, 2009).

We have to think about time all the time. We trade time for money. Efficient use of time is namely one of the greatest indicators of competitiveness.

2. Role and impact of scheduling

Scheduling is a decision-making process used on a regular basis in many manufacturing and service industries. These forms of decision-making play an important role in procurement and production, in transportation and distribution, and in information processing and communication (Blazewicz et al., 2007). The scheduling functions in a company rely on mathematical techniques and heuristic methods to allocate limited resources to the activities that have to be done. This allocation of resources has to be done in such a way that the company optimizes its objectives and achieves its goals. Resources may be machines in a workshop, crews at a construction site, or processing units in a computing environment. Activities may be operations in a workshop, stages in a construction project, or computer programs that have to be executed. Each activity may have a priority level, an earliest possible starting time and a due date. Objectives can take many different forms, such as minimizing the time to complete all activities, minimizing the number of activities that are completed after the committed due dates, and so on (Singh, 2014).

Within scheduling in manufacturing in the chapter, a generic manufacturing environment and the role of its scheduling function will be described. Orders that are released in a manufacturing setting have to be translated into jobs with associated due dates. These jobs often have to be processed on the machines in a work-station in a given order or sequence. The processing of jobs may sometimes be delayed if certain machines are busy. Preemptions may occur when high priority jobs are released which have to be processed at once. Unexpected events on the shop-floor, such as machine breakdowns or longer-than-expected processing times, also have to be taken into account, since they may have a major impact on the schedules. Developing, in such an environment, a detailed schedule of the jobs to be performed helps maintain efficiency and control of operations (Kaban et al., 2012).

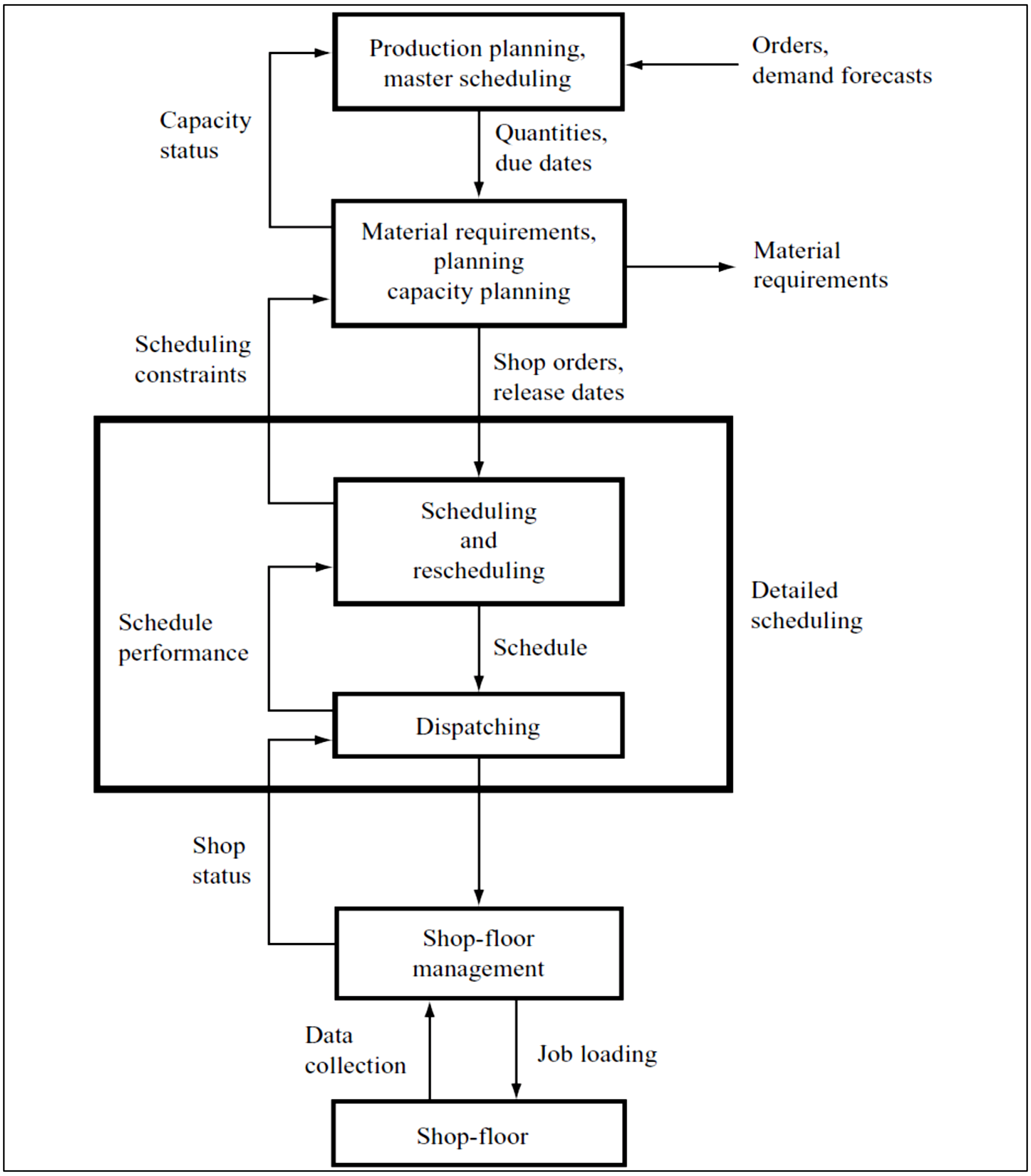

Fig. 1. Information flow diagram in a manufacturing system (Pinedo, 2005)

The scheduling process also interacts with the production planning process, which handles medium-term to long-term planning for the entire organization. This process intends to optimize the firm’s overall product mix and long-term resource allocation based on inventory levels, demand forecasts and resource requirements (Velaga, 2013). Decisions made at this higher planning level may impact the more detailed scheduling process directly. Fig. 1 shows the information flow in a manufacturing system.

3. Advanced scheduling methods

3.1 Filtered beam search

This method is based on the ideas of branch and bound. Enumerative branch and bound methods are currently the most widely used methods for obtaining optimal solutions to "NP-hard" scheduling problems. The main disadvantage of branch and bound is that it is usually extremely time consuming, because the number of nodes one must consider is very large (Fig. 2).

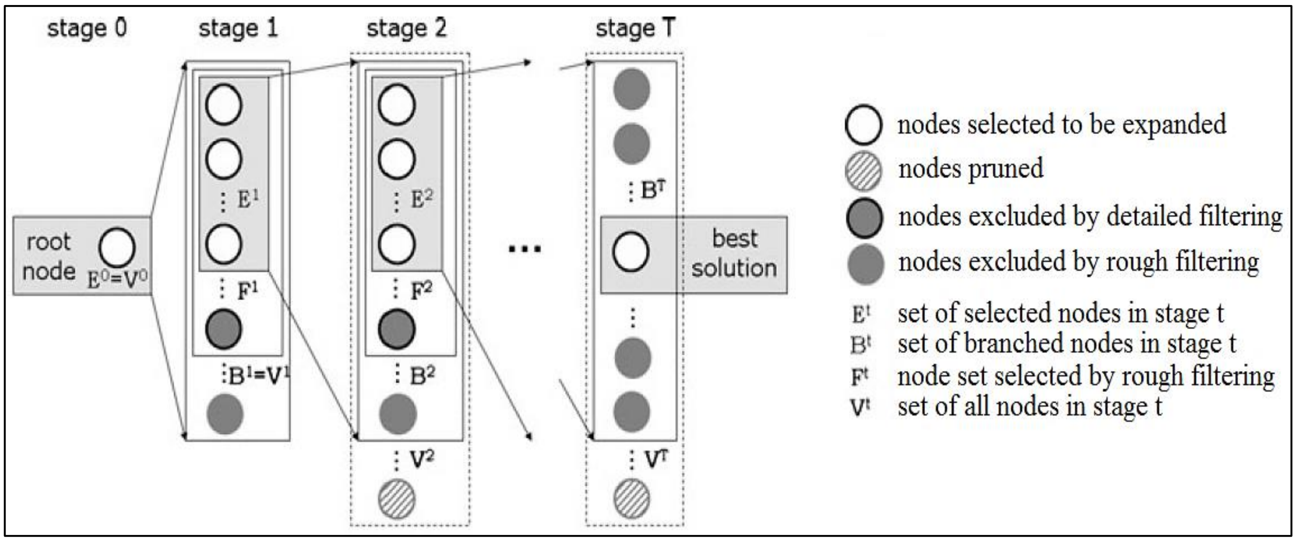
stage 0
stage 1
stage 2
stage T
root node $E^0=V^0$
E^1
F^1
$B^1=V^1$
E^2
F^2
B^2
V^2
B^T
best solution
V^T
nodes selected to be expanded
nodes pruned
nodes excluded by detailed filtering
nodes excluded by rough filtering
E^t set of selected nodes in stage t
B^t set of branched nodes in stage t
F^t node set selected by rough filtering
V^t set of all nodes in stage t

Fig. 2. Schematic representation of Filtered Beam Search

Consider, for example, a single machine problem with *n* jobs. Assume that for each node at level *k*, jobs have been selected for the first *k* positions. There is a single node at level 0, with *n* branches emanating from it to *n* nodes at level 1. Each node at level 1 branches out into *n*–1 nodes at level 2, resulting in a total of $n \cdot (n-1)$ nodes at level 2. At level *k*, there are $n!/(n-k)!$ nodes. At the bottom level, level *n*, there are *n*! nodes. Branch and bound method attempts to eliminate a node by determining a lower bound on the objective for all partial schedules that sprout out of that node. If the lower bound is higher than the value of the objective under a known schedule, then the node may be eliminated and its offspring disregarded. If one could obtain a reasonably good schedule through some clever heuristic before going through the branch and bound procedure, then it might be possible to eliminate many nodes. Even after these eliminations, there are usually still too many nodes to be evaluated. The main advantage of branch and bound is that, after evaluating all nodes, the final solution is known with certainty to be optimal (Tasic et al., 2007).

Filtered beam search is an adaptation of branch and bound in which not all nodes at any given level are evaluated. Only the most promising nodes at level *k* are selected as nodes to branch from. The remaining nodes at that level are discarded permanently. The number of nodes retained is called the beam width of the search. The evaluation process that determines which nodes are the promising ones is a crucial component of this method. Evaluating each node carefully, to obtain an estimate for the potential of its offspring, is time consuming. There is a trade-off here: a crude prediction is quick but may lead to discarding good solutions, whereas a more thorough evaluation may be prohibitively time consuming. Here is where the filter comes in. For all the nodes generated at level *k*, a crude prediction is done. Based on the outcome of these crude predictions, a number of nodes are selected for a thorough evaluation, and the remaining nodes are discarded permanently. The number of nodes selected for a thorough evaluation is referred to as the filter width. Based on the outcome of the careful evaluation of all nodes that pass the filter, a subset of these nodes (the number being equal to the beam width, which, therefore, cannot be greater than the filter width) is selected, from which further branches will be generated.

Example 1: Consider the instance of $1 \mid \mid \sum w_j T_j$ (using notation $\alpha \mid \beta \mid \gamma$). The objective is to minimize the sum of the weighted tardinesses. Table 1 contains the basic data (all jobs are available at time zero; due dates are extremely low to expose the objective function).

Job *j*	1	2	3	4	*j* – job number
p_j	10	10	13	4	p_j – processing time of job
d_j	4	2	1	12	d_j – due date of job
w_j	14	12	1	12	w_j – weight of job

Tab. 1. Data for four jobs and notation

Because the number of jobs is rather small, only one type of prediction is made for the nodes at any particular level. No filtering mechanism is used. The beam width is chosen to be 2, which implies that at each level only two nodes are retained. The prediction at a node is made by scheduling the unscheduled jobs according to the ATC (Apparent Tardiness Cost) rule. With the due-date range factor:

$$R = \frac{d_{\max} - d_{\min}}{\hat{C}_{\max}} \tag{1}$$

R = 11/37 and the due-date tightness factor:

$$\tau = 1 - \frac{\bar{d}}{\hat{C}_{\max}} \tag{2}$$

$\tau \approx 32/37$, the look-ahead parameter *k*, estimated by the following equations:

$$\begin{aligned} k &= 4.5 + R, \quad R < 0.5 \\ k &= 6 - 2 \cdot R, \quad R \geq 0.5 \end{aligned} \tag{3}$$

is chosen to be 5.

A branch and bound tree is constructed with the assumption that the sequence is developed, starting from t = 0. So, at the j^{th} level of the tree jobs are put into the j^{th} position. At the level 1 of the tree, there are four nodes: (1,*,*,*), (2,*,*,*), (3,*,*,*) and (4,*,*,*), see Fig. 3. Using the ATC rule to the remaining jobs at each one of four nodes results in four sequences: (1,4,2,3), (2,4,1,3), (3,4,1,2) and (4,1,2,3) with objective values 408, 436, 814 and 440. Because the beam width is 2, only the first two nodes are retained.

Each of these two nodes leads to three nodes at level 2. Node (1,*,*,*) leads to nodes (1,2,*,*), (1,3,*,*) and (1,4,*,*), and node (2,*,*,*) leads to nodes (2,1,*,*), (2,3,*,*) and (2,4,*,*). Applying the ATC rule to the remaining two jobs in each one of the six nodes at level 2 results in nodes (1,4,*,*) and (2,4,*,*) being retained and the remaining four being discarded.

Two nodes at level 2 lead to four nodes at level 3 (the last level), (1,4,3,2), (1,4,2,3), (2,4,1,3) and (2,4,3,1). Of these four sequences, sequence (1,4,2,3) is the best with a total weighted tardiness equal to 408. This sequence is optimal.

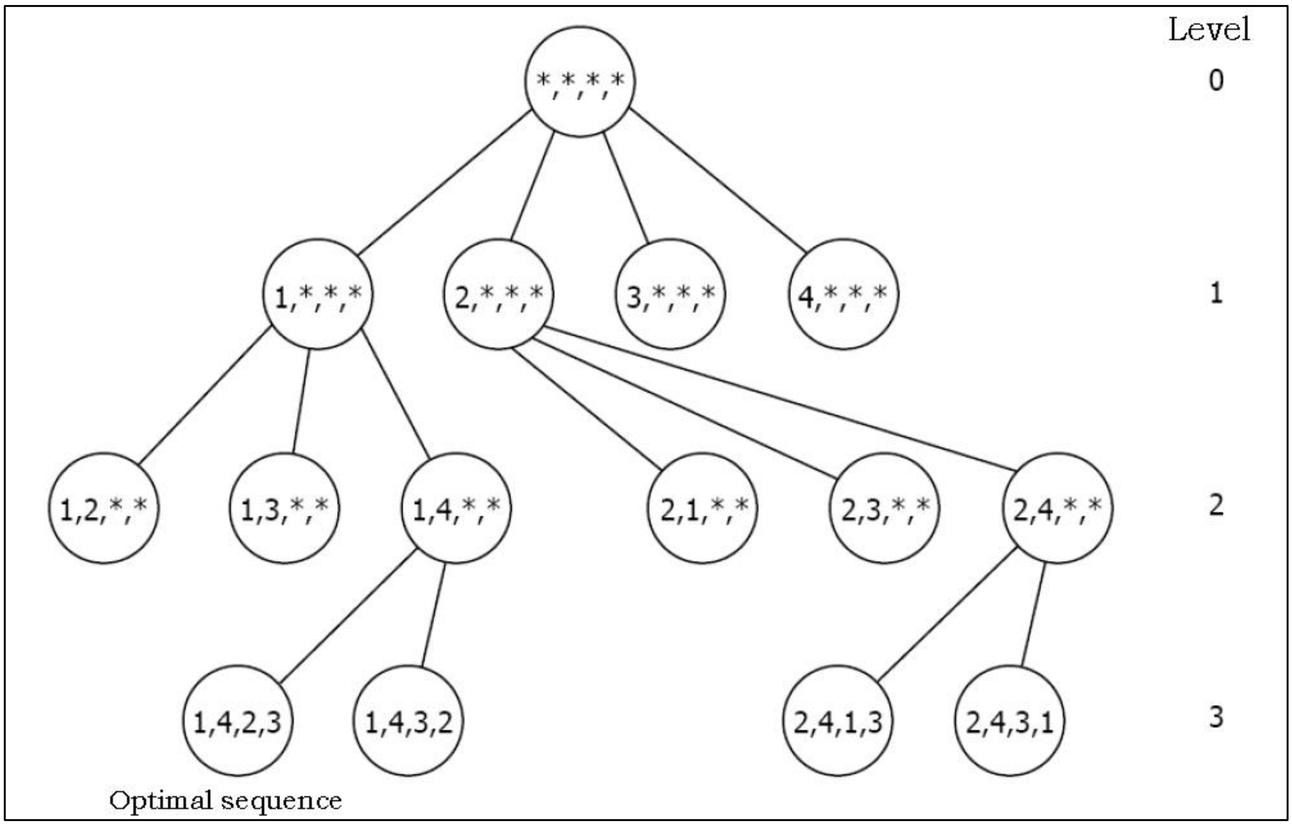

Fig. 3. Beam search applied to $1||\sum w_j T_j$

3.2 Constraint-guided heuristic search

In many real-world situations, there is not really an objective. Rather it is required only to generate a feasible schedule that satisfies various constraints and rules. One approach for generating schedules in these situations is referred to in the literature (Pinedo, 2005) as constraint-guided search. This approach has been very popular among computer scientists and artificial intelligence experts.

Constraint-guided search may be described best through an example. Consider a number of not necessarily identical machines in parallel. A job has to be processed only on the one of the machines; for each job there may be a feasible set of machines M_j to choose from. Job j requires a processing time $p_j = 1, j = 1, \ldots, n$ and has release date r_j and the due date d_j. The goal is to find a feasible schedule in which all jobs are processed within their respective time windows. In this case the optimal schedule has an objective of value 0 (perfect feasibility).

Constraint-guided search may operate according to the following rules. Jobs are scheduled one at a time. When a job is scheduled, it is assigned to a specific time slot on a specific machine, which is still free. At the each iteration, an unassigned job is selected according to a set of job rules that have been arranged in some priority. The job rules specify whether the particular job actually can be processed on a given machine.

The jobs can be ordered according to their criticality or flexibility; the job with the least flexibility is the most critical and has the highest priority. The flexibility of a job can be measured in several ways (flexibility in time – slack time, flexibility with regard to the number of appropriate machines, etc.).

The machines also can be ordered in such a way that the machine with the least flexibility has the highest priority (the flexibility of a machine, measured by the number of jobs that can be processed on the machine).

An important concept in constraint-guided search is constraint propagation. The assignment of a particular job to a given time slot on a given machine has implications with regard to the assignment of other jobs on the given machine and on other machines. These implications may point to the violation of hard constraints and may indicate that the associated part of the search space can be disregarded.

Example 2: Consider the problem $P3 \mid r_j, p_j = 1, M_j \mid \sum U_j$. $U_j = 1$, if $C_j > d_j$, otherwise $U_j = 0$. We have three machines and nine jobs. The processing times, release dates, and due dates of the job are presented in Table 2. If the processing time of a job on a machine is infinity (∞), then the job cannot be processed on that machine. The goal is to find a feasible schedule with all jobs completed on time ($\Sigma\ U_j = 0$).

Job j	1	2	3	4	5	6	7	8	9
p_{1j}	∞	1	1	∞	1	∞	∞	1	1
p_{2j}	1	1	∞	1	1	1	∞	1	1
p_{3j}	1	1	∞	1	1	∞	1	∞	1
r_j	1	1	0	0	0	0	1	1	2
d_j	3	2	1	2	1	1	3	3	3

Tab. 2. Data for nine jobs

There are nine timeslots and three on each machine. Each job has its own time window, which represents a set of constraints. For each job, a flexibility factor Φ_j can be calculated. In this example: Φ_j is the number of timeslots to which a job may be assigned on various machines. The flexibility factor Φ_j of job j is presented in Table 3.

Instead of the flexibility of a machine, the flexibility of a timeslot on a machine is determined. It is defined as the number of jobs that can be processed during the timeslot. The flexibility factor $\Phi_{(i,l)}$ of job timeslot (i, l) is presented in Table 4.

Job j	1	2	3	4	5	6	7	8	9
Φ_j	4	3	1	4	3	1	2	4	3

Tab. 3. Flexibility factor of nine jobs

Machine	1	1	1	2	2	2	3	3	3
Timeslot	(1, 1)	(1, 2)	(1, 3)	(2, 1)	(2, 2)	(2, 3)	(3, 1)	(3, 2)	(3, 3)
$\Phi_{(i,l)}$	2	2	2	3	4	3	2	4	3

Tab. 4. Flexibility factor of job timeslot

Sequence of jobs in increasing flexibility results in: 3, 6, 7, 2, 5, 9, 1, 8, 4. Priorities of the timeslots may result in the sequence (1, 3), (3, 1), (1, 1), (1, 2), (2, 1), (2, 3), (3, 3), (3, 2), (2, 2).

Job 3 is selected first. It is checked to determine whether it is allowed to be processed in the timeslot (1, 3). It is not (too late). The next timeslot is tried (3, 1) – not on 3rd machine, and so on, until a timeslot is found during which it is allowed to be processed. Job 3 is then assigned to slot (1, 1). Job 6 is considered in the same manner and assigned to slot (2, 1). Continuing in this manner results in the following assignment – see Table 5. All the constraints are fulfilled.

Timeslot	(1, 1)	(1, 2)	(1, 3)	(2, 1)	(2, 2)	(2, 3)	(3, 1)	(3, 2)	(3, 3)
Job	3	2	9	6	8	1	5	4	7

Tab. 5. Assignment of jobs to timeslots

After a job has been assigned, the flexibility factors of the remaining jobs to be assigned and the remaining timeslots available may change. It would have been possible to reorder the remaining jobs, as well as the remaining timeslots, based on the new flexibility factors. In the example above this was not done.

Constraint-guided search does not always yield a feasible solution after the first pass. It may occur that when the last job has to be assigned, no feasible assignment is possible. In this case, the method has to rely on a postprocessing procedure, which, through pairwise interchanges, attempts to construct a feasible solution.

3.3 Genetic algorithms

Genetic algorithms (GA) are an optimization methodology based on a direct analogy to Darwinian natural selection and mutations in biological reproduction. In principle, genetic algorithms encode a parallel search through concept space, with each process attempting coarse-grain hill climbing. Instances of a concept correspond to individuals of a species. Induced changes and recombinations of these concepts are tested against an evaluation function to see which ones will survive to the next generation (Gao et al., 2007). The use of genetic algorithms (Fig. 4) requires five components:

1. A way of encoding solutions to the problem – fixed length string of symbols.

2. An evaluation function that returns a rating for each solution.

3. A way of initializing the population of solutions.
4. Operators that may be applied to parents when they reproduce to alter their genetic composition such as crossover (i.e., exchanging a randomly selected segment between parents), mutation (i.e., gene modification), and other domain specific operators.
5. Parameter setting for the algorithm, the operators, and so forth.

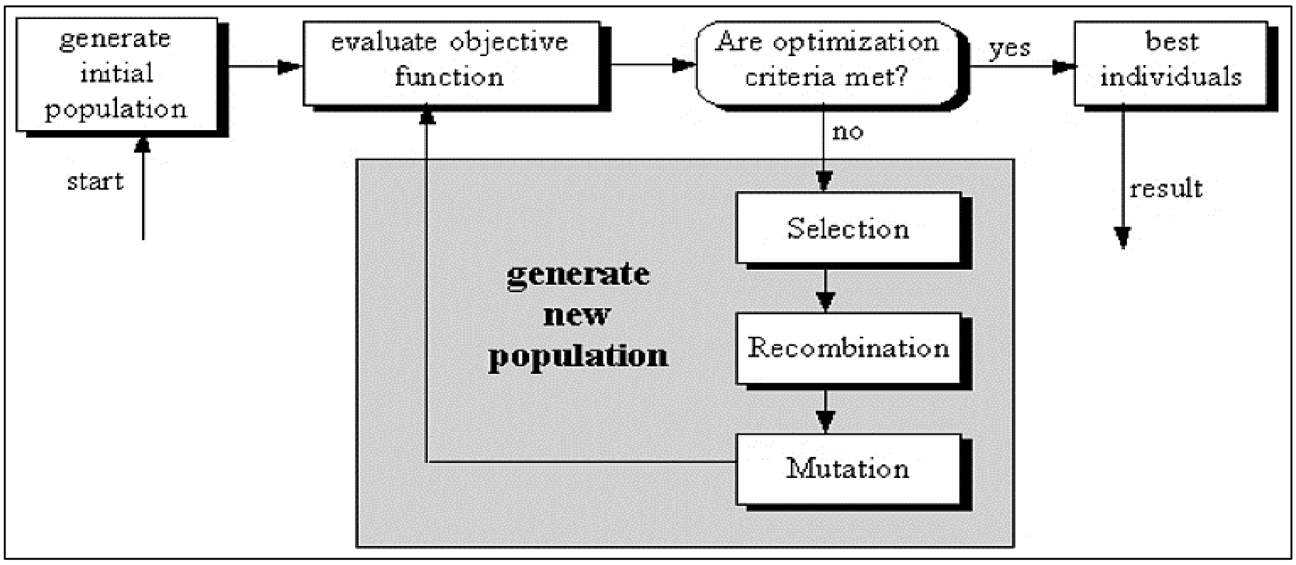

Fig. 4. Structure of a simple genetic algorithm

Two basic encoding approaches, direct and indirect, are applicable. The direct approach encodes a job shop schedule as a chromosome and the genetic operators are used to evolve these chromosomes into better schedules. In the indirect approach, a sequence of decision preferences is encoded into the chromosome, for example scheduling rules for job assignments, and the genetic operators are applied to improve the ordering of the various preferences. A job shop schedule is then generated from the sequence of preferences (Nagano et al., 2008).

A number of approaches have been utilized in the application of genetic algorithms (GA) to scheduling problems:
1. Genetic algorithms with blind recombination operators have been utilized in job shop scheduling. Their emphasis on relative ordering schema, absolute ordering schema, cycles, and edges in the offsprings will arise differences in such blind recombination operators.
2. Sequencing problems have been also addressed by the mapping of their constraints to a Boolean satisfiability problem using partial payoff schemes. This scheme has produced good results for very simple problems.
3. Heuristic genetic algorithms have been applied to job shop scheduling. In these genetic schemes, problem specific heuristics are incorporated in the recombination operators (such as optimization operators based).

Example 3: A simple genetic algorithm is used to treat the job shop problem. This algorithm was developed independently, without regard for the work of other researchers (Lestan et al., 2009). The intention was to make a simple algorithm, which will try to find the schedule with the smallest makespan. Only genetic operations are used in order to achieve this. It is possible to schedule a various number of jobs, but neither release times nor due dates are considered. Only selection and permutation are used as genetic operations. The algorithm uses random moves to search for the optimal schedule in the solution space, which means that the solution is obtained without the help of heuristic methods.

How to encode solutions to chromosomes to ensure feasible solutions is a key issue for genetic algorithms. In our algorithm, the preference list-based representation is used. In this encoding method, the operations are arranged in a certain order. It depends on this order, how the operations will be processed on the machines. It is very important, that the precedence constrains of operations of individual jobs are considered. This means, that the sequence of operations of a job must stay intact also in the encoded solution. How the encoding works is shown on a simple example. Table 6 shows a 3 × 3 instance; 3 jobs (9 operations) must be scheduled on 3 machines (M1, M2, M3) to achieve the smallest possible makespan.

Jobs	**Processing times**			**Processing order**		
	Operations			Operations		
	1	2	3	1	2	3
J1	29	78	9	machine M1	machine M2	machine M3
J2	43	90	28	machine M1	machine M3	machine M2
J3	91	85	74	machine M2	machine M1	machine M3

Tab. 6. Data for a 3 × 3 instance

From the Table 6 it is possible to write the operation sequence for each job:

J1 (J1 M1 29) (J1 M2 78) (J1 M3 9)
J2 (J2 M1 43) (J2 M3 90) (J2 M2 28)
J3 (J3 M2 91) (J3 M1 85) (J3 M3 74)

Job J1 must first be processed on machine M1 for 29 time units. After that on machine M2 for 78 units and the last is machine M3 for 9 units. Similar goes for the other two jobs. The schedule for this instance is encoded into a string, where the position of the operation in the string plays an important role. Operations are ordered with the help of a randomizer. Therefore, it is important to use a reliable randomizer. String making in our case looks like this:

a) A list of first operations of all jobs has been made.

((J1 M1 29) (J2 M1 43) (J3 M2 91))

b) From the list of first operations, one operation is chosen randomly; let's say (J2 M1 43). This operation is the first operation in the string. The operation is taken from the corresponding job and inserted into the string.

J1 (J1 M1 29) (J1 M2 78) (J1 M3 9)
J2 (J2 M3 90) (J2 M2 28)
J3 (J3 M2 91) (J3 M1 85) (J3 M3 74)

String:
((J2 M1 43))

c) Again, a list of first operations of all the jobs is made and an operation is randomly selected from the list; let’s say (J1 M1 29). This operation is taken from the corresponding job and inserted as second operation in the string.

J1 (J1 M2 78) (J1 M3 9)
J2 (J2 M3 90) (J2 M2 28)
J3 (J3 M2 91) (J3 M1 85) (J3 M3 74)

String:

((J2 M1 43) (J1 M1 29))

d) The procedure is repeated until all the operations from the jobs are transferred into the string.

If the procedure, described above, would be continued until the end, the string could look like this:

((J2 M1 43) (J1 M1 29) (J1 M2 78) (J3 M2 91) (J2 M3 90)
(J3 M1 85) (J2 M2 28) (J1 M3 9) (J3 M3 74))

This string will be used later on for demonstrations. A closer look at the string reveals that the precedence constrains have been considered during the making of the string. Job operations in the string still have the same processing order, but now they are mixed together. Why this is so important is explained below.

Because the string making is left to coincidence, it is possible to make many versatile strings (organisms) which are necessary for the initial population.

Only feasible strings represent a solution to our problem and therefore it is very important that feasibility is maintained throughout the searching process. Because in our case the goal lies in the time optimization of schedules, we are interested in the makespan. Besides the makespan, we are interested also in the processing order on the machines. The Gantt chart (string evaluation) is done step by step with adding operations one after another. In our case, the operations are added directly from the string, from the left side to the right side. The operations, which are at the beginning of the string, have a higher processing priority than those at the end. This means, that the Gantt chart and the makespan depend only upon the order in the string. That is why it is so important, that the operation order in the string is according to the precedence constrains. Otherwise, the evaluation would give a false value. The Gantt chart for our string is shown in Fig. 5.

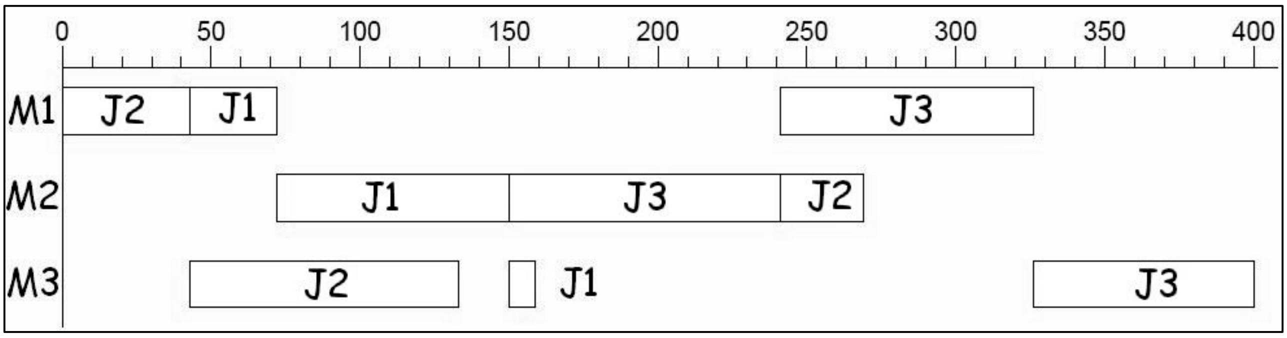

Fig. 5. Gantt chart for the 3 × 3 instance

String:

((J2 M1 43) (J1 M1 29) (J1 M2 78) (J3 M2 91) (J2 M3 90)
(J3 M1 85) (J2 M2 28) (J1 M3 9) (J3 M3 74))

Because the use of graphic Gantt charts in programming would be annoying, Gantt charts in numerical form are used. These charts are not as synoptic as graphical, but it is possible to comprehend all the important data from them. The Gantt chart in Fig. 4 looks in numerical form like this:

M1 (0 J2 43) (43 J1 72) (241 J3 326)
M2 (72 J1 150) (150 J3 241) (241 J2 269)
M3 (43 J2 133) (150 J1 159) (326 J3 400)

Genetic operations are the driving force in genetic algorithms. Which operations are reasonable to use for solving a certain problem depends on the encoding method. The only genetic operation, which is independent from encoding, is selection. Selection is the most simple of all genetic operations. In our algorithm, the tournament selection is used. Its purpose is to maintain the core of good solutions intact. This is done with transferring good solutions from one generation to the next one. Because selection does not change the organism (string), we have no problem with maintaining feasibility, which is not the case in all other genetic operations. The use of the crossover operation can be problematical in some cases. The crossover operation often produces infeasible offspring, which are difficult to repair.

The only genetic operation besides selection, which is used in our algorithm, is permutation. The permutation is based on switching operations inside the organism. The organism, which will be permutated, is chosen with the selection. When executing the permutation it is necessary to consider the precedence constrains. The permutation procedure is described and shown on our string from Table 6:

a) A random operation is chosen from the string; let’s say (J2 M1 43).

String:

((J2 M1 43) (J1 M1 29) (J1 M2 78) (J3 M2 91) (J2 M3 90)
(J3 M1 85) (J2 M2 28) (J1 M3 9) (J3 M3 74))

b) The left and the right border for the chosen operation have to be defined. Because the chosen operation belongs to job J2, it is necessary to search for the first operation, left and right of the chosen operation, which belongs to job J2. If the operation does not exist, the border is represented by the end or the beginning of the string. In our case, the right border is represented by the operation (J2 M3 90) and the left border is presented by the beginning of the string. In the string, the space between the borders is marked with square brackets.

String:

([(J2 M1 43) (J1 M1 29) (J1 M2 78) (J3 M2 91)] (J2 M3 90)
(J3 M1 85) (J2 M2 28) (J1 M3 9) (J3 M3 74))

c) A random position between the brackets is chosen where the operation (J2 M1 43) is inserted; let us say in front of (J1 M2 78).

String:

((J1 M1 29) (J2 M1 43) (J1 M2 78) (J3 M2 91) (J2 M3 90)
(J3 M1 85) (J2 M2 28) (J1 M3 9) (J3 M3 74))

This procedure randomly changes the chosen organism, which also changes the solution, which the organism represents. Because there is often necessary to switch more than one operation in the organism, it is possible to repeat the whole procedure over and over.

Evolution parameters are parameters, which influence the searching procedure of the genetic algorithm. If we want to obtain good solutions, the parameters have to be set wisely. These parameters are:

- selection pressure,
- population size,
- amount of change, made by genetic operation,
- share of individual genetic operations in the next generation,
- number of generations (stopping criterion),
- number of independent civilizations.

The selection pressure defines what kind of solutions will be used in genetic operations. If the selection pressure is high, then only the best solutions will get the chance. If it is low, then also worse solutions are used. The higher the selection pressure, the higher the possibility that the search will end up in a local optimum. If it is too low, the search procedure examines insignificant solutions, which protracts the whole search.

When an organism is being modified, it is necessary to specify how much the genetic operation will change the organism. It is recommended that small and large changes are made to the organisms. This assures versatility in the population.

Each genetic operation must make a certain amount of organisms for the next generation. At least 10 % of the next population has to be made with selection (Brezocnik, 2000), so that the core of good solutions is maintained. All the other organisms are created with other genetic operations.

If the number of generations is multiplied with the size of the population, we get the number of organisms, which have been examined during the search. When solving complex problems, the number of organisms is greater than in easier cases. The question is, should the search be executed with a small population and a lot of generations or opposite. In most cases, a compromise is the best choice.

Because the search with genetic algorithms bases on random events, it is not necessary that good solutions are obtained in every civilization. In most cases, the search stops in a local optimum. That is why it is necessary to execute multiple searches for a given problem.

The algorithm was tested on the benchmark 10 × 10 instance (see the results in Table 7 for the data in Table 8), which was proposed by Fisher and Thompson (1963).

Rows contain the order of the operations for each job J_i: each entry (M*j*, *p*) contains the code of machine M_j and the processing time p_{ij} on it.

Parameter	**10 × 10**
Population size	200
Number of generations	500
Number of independent populations	100
Optimal solution	930
Best solution obtained	941

Tab. 7. Evolution parameters and results for the 10 × 10 instance

Jobs	**Operation sequence**									
	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
J1	M1 29	M2 78	M3 9	M4 36	M5 49	M6 11	M7 62	M8 56	M9 44	M10 21
J2	M1 43	M3 90	M5 75	M10 11	M4 69	M2 28	M7 46	M6 46	M8 72	M9 30
J3	M2 91	M1 85	M4 39	M3 74	M9 90	M6 10	M8 12	M7 89	M10 45	M5 33
J4	M2 81	M3 95	M1 71	M5 99	M7 9	M9 52	M8 85	M4 98	M10 22	M6 43
J5	M3 14	M1 6	M2 22	M6 61	M4 26	M5 69	M9 21	M8 49	M10 72	M7 53
J6	M3 84	M2 2	M6 52	M4 95	M9 48	M10 72	M1 47	M7 65	M5 6	M8 25
J7	M2 46	M1 37	M4 61	M3 13	M7 32	M6 21	M10 32	M9 89	M8 30	M5 55
J8	M3 31	M1 86	M2 46	M6 74	M5 32	M7 88	M9 19	M10 48	M8 36	M4 79
J9	M1 76	M2 69	M4 76	M6 51	M3 85	M10 11	M7 40	M8 89	M5 26	M9 74
J10	M2 85	M1 13	M3 61	M7 7	M9 64	M10 76	M6 47	M4 52	M5 90	M8 45

Tab. 8. The 10 job 10 machine instance (Fisher and Thompson, 1963)

As it can be seen from Table 7, the algorithm did not manage to find the optimal solution for the problem, but due its simplicity, the algorithm was able to obtain good solutions.

The number of possible schedules *S* (solutions) can be calculated with eq. (4), where *m* represents the number of machines and *n* represents the number of jobs.

$$S = (n!)^m \tag{4}$$

So, for our case we get: $S_{10\times10} \approx 4 \cdot 10^{65}$ possible schedules.

Because the algorithm is written in the LISP programming language, the search procedure took relatively long time. The best obtained solution deviates 1.2 % from the optimal solution. The best result was obtained in only 1 run out of 100 runs.

The search procedure often falls in a local optimum. When this happens, it is very unlikely that the search will proceed to a better solution, because a memory function is not present. This means that the quality of the final solution depends on the initial population and pure chance. So, the search procedure must be repeated several times for a specific problem (Tay & Ho, 2008).

4. Conclusion

Scheduling in praxis means knowing:

- The status and priority of each order on the shop floor.
- Which machines and other resources (e.g. sub components, materials, tooling and operators) are required for each order.
- When these resources are required and when they will be available. When do we expect the resources to arrive or be available?

Scheduling balances due dates, machine capacity, tooling and labour to develop a realistic plan of action to move orders through various operation steps (Saravanan et al., 2008).

Advanced scheduling methods are potential tools for making (near) optimal and feasible schedules (Jarboui, 2008). For the applications in praxis, it is important to understand how the resulting schedule is generated, unless the schedule will not be used. It is hard to understand why operations are prioritised in a certain sequence for the operators that only see a dispatch list. The resulting sequence of operations has to take into account all capacity, inventory, material availability and delivery time constraints and, at the same time, it should increase throughput and minimize the operational costs (Lei, 2008).

Some classes of manufacturing models, which already have been considered in the literature, may in the future be generalized and extended in new directions. Models considered in the literature often focus on a single objective. In practice, it may very well be the case that several objectives have to be considered at the same time and that the user would like to see a parametric analysis in order to evaluate the trade-offs.

5. Acknowledgement

The authors are very grateful to all the people who have contributed directly or indirectly to the completion of this chapter. Thanks are also due to the Slovenian Research Agency (ARRS).

6. References

Baesler, F.; Gatica, J. & Correa, R. (2015). Simulation optimisation for operating room scheduling. *International Journal of Simulation Modelling*, Vol. 14, No. 2, 215-226, ISSN 1726-4529

Blazewicz, J.; Ecker, K. H.; Pesch, E.; Schmidt, G. & Weglarz, J. (2007). *Handbook on scheduling*. Springer-Verlag, ISBN 978-3-540-28046-0, Berlin

Brezocnik, M. (2000). *The use of genetic programming in intelligent manufacturing systems* (in Slovene). Faculty of Mechanical Engineering, University of Maribor, ISBN 86-4350306-1, Maribor

Buchmeister, B.; Kremljak, Z.; Pandza, K. & Polajnar, A. (2004). Simulation study on the performance analysis of various sequencing rules. *International Journal of Simulation Modelling*, Vol. 3, No. 2-3, 80-89, ISSN 1726-4529

Fisher, H. & Thompson, G. L. (1963). Probabilistic learning combinations of local jobshop scheduling rules, In: *Industrial Scheduling*, Muth, J. F.; Thompson, G. L. (Eds.), 225-251, Prentice Hall, Englewood Cliffs

Gao, J.; Gen, M.; Sun, L. & Zhao, X. (2007). A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, Vol. 53, No. 1, 149-162, ISSN 0360-8352

Huang, R. (2010). Multi-objective job-shop scheduling with lot-splitting production. *International Journal of Production Economics*, Vol. 124, No. 1, 206-213, ISSN 0925-5273

Jarboui, B.; Ibrahim, S.; Siarry, P. & Rebai, A. (2008). A combinatorial particle swarm optimization for solving permutation flowshop problems. *Computers & Industrial Engineering*, Vol. 54, No. 3, 526-538, ISSN 0360-8352

Kaban, A. K.; Othman, Z. & Rohmah, D. S. (2012). Comparison of dispatching rules in job-shop scheduling problem using simulation: a case study. *International Journal of Simulation Modelling*, Vol. 11, No. 3, 129-140, ISSN 1726-4529

Lei, D. (2008). A Pareto archive particle swarm optimization for multi-objective job-shop scheduling. *Computers & Industrial Engineering*, Vol. 54, No. 4, 960-971, ISSN 0360-8352

Lestan, Z.; Brezocnik, M.; Buchmeister, B.; Brezovnik, S. & Balic, J. (2009). Solving the job-shop scheduling problem with a simple genetic algorithm. *International Journal of Simulation Modelling*, Vol. 8, No. 4, 197-205, ISSN 1726-4529

Nagano, M. S.; Ruiz, R. & Lorena, L. A. N. (2008). A constructive genetic algorithm for permutation flowshop scheduling. *Computers & Industrial Engineering*, Vol. 55, No. 1, 195-207, ISSN 0360-8352

Pinedo, M. L. (2005). *Planning and scheduling in manufacturing and services*. Springer Science+Business Media, ISBN 0-387-22198-0, New York

Saravanan, M.; Haq, A. N.; Vivekraj, A. R. & Prasad, T. (2008). Performance evaluation of the scatter search method for permutation flowshop sequencing problems. *International Journal of Advanced Manufacturing Technology*, Vol. 37, No. 11-12, 1200-1208, ISSN 0268-3768

Singh, A. (2014). Resource constrained multi-project scheduling with priority rules & analytic hierarchy process. *Procedia Engineering*, Vol. 69 (24th DAAAM International Symposium), 725-734, ISSN 1877-7058

Tasic, T.; Buchmeister, B. & Acko, B. (2007). The development of advanced methods for scheduling production processes. *Strojniski vestnik – Journal of Mechanical Engineering*, Vol. 53, No. 12, 844-857, ISSN 0039-2480

Tay, J. C. & Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, Vol. 54, No. 3, 453-473, ISSN 0360-8352

Velaga, P. (Optisol) (2013). Production Scheduling for Job Shops, Available from: *http://www.optisol.biz/job_shop_scheduling.html*, accessed on 25-11-2013

Yagmahan, B. & Yenisey, M. M. (2009). Scheduling practice and recent developments in flow shop and job shop scheduling. In: *Computational Intelligence in Flow Shop and Job Shop Scheduling*, Chakraborty, U. K. (Ed.), 261-300, Springer-Verlag, ISBN 978-3-642-02835-9, Berlin