Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Control Engineering

Effectiveness of Scheduling Algorithms Implementation for Manufacturing Processes

Thesis

Prague, 2009 Author: Jan Zahradník

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Jan Zahradník**

Studijní program: Elektrotechnika a informatika (magisterský), strukturovaný
Obor: Kybernetika a měření, blok KM1 - Řídicí technika

Název tématu: **Efektivita implementace algoritmů pro rozvrhování výrobních procesů**

Pokyny pro vypracování:

1. Seznamte se s algoritmy pro rozvrhování výrobních procesů.
2. Seznamte se s knihovnami pro operace s grafy.
3. Pro vybranou případovou studii navrhněte a implementujte algoritmus pro rozvrhování výroby. Při implementaci se zaměřte na efektivitu řešení. Vámi navržený algoritmus srovnejte s jiným vybraným již existujícím řešením.
4. Odzkoušejte a zdokumentujte Vámi navržené řešení.

Seznam odborné literatury:

[1] Peter Brucker, Sigrid Knust. Complex Scheduling, Springer Berlin Heidelberg, 2006.
[2] Peter Brucker, Silvia Heitmann, Johann Hurink and Tim Nieberg, Job-shop scheduling with limited capacity buffers, OR Spectrum, Springer Berlin / Heidelberg, 2006.
[3] Birger Franck, Klaus Neumann and Christoph Schwindt, Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling, OR Spectrum, Springer Berlin / Heidelberg, 2001.

Vedoucí: Ing. Přemysl Šůcha, Ph.D.

Platnost zadání: do konce zimního semestru 2009/10

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 27. 2.. 2009

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze, dne 18.5.2009 .....

podpis

Acknowledgements

I would like to thank Přemysl Šůcha, who was my supervisor. He provided me with many helpful suggestions, important advice and constant encouragement during the course of this work. My special appreciation goes to my parents, who always kept me away from family responsibilities and encouraged me to concentrate on my study. This work was supported by project CEPOT (http://www.cepot.cz) and Department of Control Engineering, CTU FEL in Prague.

Abstrakt

Tato práce se zabývá efektivitou algoritmů pro rozvrhování výrobních procesů. Efektivita implementace rozvhrovacích algoritmů je zkoumána z hlediska výběru programovacího jazyka, možnosti použití softwarových knihoven pro práci s grafy a s maticemi, a shrnutím obecných zásad pro psaní efektivního, časově náročného algoritmu. V průběhu práce byly naprogramovány dva rozvrhovací algoritmy. U těchto algoritmů je zkoumána možnost úpravy pro řešení jiné cílové funkce. Implementované algoritmy jsou testovány pro obecné i reálné rozvrhovací problémy. Jejich výsledky jsou porovnány s výsledky jiných algoritmů publikovaných odbornou veřejností.

Abstract

This work deals with efficiency of algorithms for scheduling in manufacturing. Efficiency of implementation of scheduling algorithms is investigated from the points of view e.g. selection of programming language, possibility of usage of software libraries for work with graphs and matrices, and the general recommendations for evolution of efficient, time consuming algorithms. Two scheduling algorithms were implemented in this work. Furthermore, a possibility of adaptation of algorithms for problems with another objective function is studied by these algorithms. Implemented algorithms are tested on general and real scheduling problems. Their results are compared with results of different algorithms published by science public.

Contents

1. Introduction

The scheduling deals with assigning of tasks to relevant machines (processors) so that a final schedule satisfies a given constraints. There are optimization methods which result is a time plan (schedule) determinative time points when and on which processor should be given tasks performed. A schedule is constructed with respect to given constraints and so that an objective function (a criterion) is minimized.

The objective function can be, for example, schedule length (a makespan), maximum lateness, mean flow time, mean weighted tardiness/or earliest, total weighted tardiness etc. Consequently, we can make many products after the same time, reduce penalty for later or earlier delivery of products and others. If we have a good scheduling algorithm, we can exactly schedule production on longer time perspective. In other words, scheduling algorithm is intended to optimize determinate production process.

Wide scientific public are interested in this branch, international conferences are organized and scheduling is used in practice of course. These methods are primarily useful in production optimization, production costs and in transport. Concrete examples can be: lacquer production, production of rolling ingots, scheduling of school time tables, scheduling of processes for computing technique, scheduling of transport services etc.

Problem of scheduling algorithms is that they are usually very time consuming. We are often not able to find a solution in polynomial time. For this reason, different heuristic methods are suggested. Efficiency of their implementation is studied in this work. How we already said, the scheduling algorithms are very time consuming. We know from theory, that a big acceleration of hardware does not imply the same acceleration of the algorithm. It is very important which method is chosen to solve a scheduling problem, how successfully is this method implemented and under which programming environment is the algorithm developed. Following questions were examined during implementation of algorithms. Are all operations by partial calculations needed? Are all partial calculations needed? Is it not advantageous some data counted contrariwise only once? Etc. Since the algorithms are predominantly heuristic algorithms, it is necessary to know what influence input parameters have to results. This work tries to answer on the problems described above.

Project Scheduling (RCPSP)

The resource-constrained project scheduling problem (RCPSP) (Brucker and Knust 2005) is a very general scheduling problem which may be used to model many applications in practice. The objective is to schedule some tasks (activities) over time such that scarce resource capacities are respected and a certain objective function is optimized. Examples for resources may be machines, people, rooms, money or energy, which are only available with limited capacities. As objective functions e.g. the project duration, the deviation from deadlines or costs concerning resources may be minimized.

The RCPSP may be formulated as follows. Given are n tasks $\mathrm{T} = \{T_1, T_2, \ldots, T_n\}$ and m renewable resources (processors) $R = \{r_1, r_2, \ldots r_m\}$. A constant amount of t_k units of resource R_k is available at any time. Task T_i must be processed for p_i time units. During this time period a constant amount of r_{ik} units of resource R_k is occupied. All parameter are assumed to be integers.

RCPSP/max

The RCPSP/max (Cicirello and Smith 2004, Smith and Pyle 2004) problem can be defined formally as follows. Define $P = (I, C, R)$ as an instance of RCPSP/max. Let T be the set of tasks $\mathrm{T} = \{T_0, T_1, \ldots, T_n, T_{n+1}\}$. Task T_0 is a dummy task representing the start of the project and T_{n+1} is similarly the project end. Each task T_k has a fixed duration p_k, a start-time s_k and a completion-time C_k which satisfy the constraint $s_k + p_k = C_k$. Let C be a set of temporal constraints between task pairs $\langle T_j, T_k \rangle$ of the form $s_k - s_j \in \left[D_{jk}^{min}, D_{jk}^{max}\right]$. The C are generalized precedence relations between tasks. The D_{jk}^{min} and D_{jk}^{max} are minimum and maximum time-lags between the start times of pairs of tasks. Let R be the set of renewable resources $R = \{r_1, r_2, \ldots r_m\}$. Each resource r_k has an integer capacity $c_k \geq 1$. Execution of a task T_j requires one or more resources. For each resource r_k, the task T_j requires an integer capacity $rc_{j,k}$ for the duration of its execution. An assignment of start-times to the tasks in T is time-feasible if all temporal constraints are satisfied and is resource-feasible if all resource constraints are satisfied. A schedule is feasible if both sets of constraints are satisfied. The problem is then to find a feasible schedule with minimum make span C_{max} where $C_{max}(S) = max\{C_i\}$. We wish to find a set of assignments to S such that $S_{sol} = arg\,min_s\, C_{max}(S)$. The maximum time-lag constraints are what makes this problem especially difficult. Particularly, due to the maximum time lag constraints, finding feasible solutions alone to this problem is NP-Hard (Bartusch, Mohring and Radermacher 1988).

1.1. Related Work

The related work for minimizing makespan and total weighted tardiness are presented in this chapter. This review gives summary of similar scheduling problems.

1.1.1. Minimizing The Makespan

We deal with resource constrained project scheduling problem with change over times and take-give resources (Hanzálek and Šůcha 2009) in this work. Many similar problems have been already proposed in literature. These methods and solutions are summarized in following paragraphs.

Most of exact algorithms are based on branch and bound technique (Brucker, Hilbic and Hurink 1999) but this approach is suitable for problem with less than 100 tasks. An overview of heuristic approaches is shown in Franck, Neumann and Schwindt (2001) where the authors compare truncated branch and bound techniques, schedule improvement procedures, priority rule methods and genetic algorithm for scheduling problems with general temporal and resource constraints. Their detailed experimental performance analysis compares the different heuristics and shows that large problem instances with up to 1000 tasks and several resources can be efficiently solved with sufficient accuracy.

A heuristic algorithm proposed in Smith (2004) combines the benefits of the “squeky wheel” optimization with an effective conflict resolution mechanism called “bulldozing”. The possibility of improving on the squeaky wheel optimization by incorporating aspects of genetic algorithm is suggested in Terada, Vo and Joslin (2006). Another heuristic algorithm Cesta et al. (2002) is based on constraint satisfaction problem solving. The algorithm is based on the intuition that the most critical conflicts to be resolved first are those involving tasks with large resource capacity requirements. A beam search heuristic is presented in Schwindt and Trautmann (2003). The basic principle is to relax the resource constraints by assuming infinite resource availability. Resulting resource conflicts are stepwise resolved by introducing precedence relationships among operations competing for the same resources. This heuristic is applied to a real scheduling problem, i.e. production of rolling ingots. This problem covers batching machines, renewable resources and changeover time. A memetic algorithm for the job-shop scheduling problem with minimal and maximal time-lags is described in Caumond et al. (2007). This problem is modelized as a non-oriented disjunctive graph and their algorithm is based on a memetic algorithm coupled with a powerful local search procedure.

Take-give resources were introduced in Hanzálek and Šůcha 2009. Similar types of resources are described in this paragraph. Scheduling with blocking operations (Mascis and Paccierelli 2002, Brucker and Kampmeyer 2008) can be seen as a subproblem of scheduling with take-give resources. Operations are blocking if they must stay on a machine after finishing when the next machine is occupied by another job. During this stay the machine is blocked for other jobs, i.e. blocking operations models the absence of storage capacity between machines. On the other hand, there is

a more general framework called reservoirs or storage resources (Laborie 2003) usually used to model limited storage capacity or inventory limits. In this framework each task can replenish or deplete certain amount of a resource but the resource assignment is not considered. Therefore this framework cannot deal for example with changeover times on this resource type required in the lacquer production problem to model mixing vessels cleaning.

1.1.2. Total Weighted Tardiness

There are many algorithms solving problems with Total weighted tardiness (definition is in Section 2). Few algorithms are described in following paragraphs.

Valente and Alves (2008) created Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups (definition of sequence-dependent setups is in Section 2). ATCS dispatching rule (Lee et al. 1997) is here used for determine priority used by beam search. The proposed beam search algorithms outperform the ATCS dispatching heuristic, but if number of tasks increases, fall the difference between these algorithms and ATCS dispatching heuristic.

Monch et al (2005) attempt to minimize total weighted tardiness on parallel batch machines with incompatible job families and unequal ready times of the jobs. They propose two different decomposition approaches. The first approach forms fixed batches, then assign these batches to the machines using a genetic algorithm, and finally sequences the batches on individual machines. The second approach first assigns jobs to machines using a genetic algorithm, then forms batches on each machine for the jobs assigned to it, and finally sequences these batches. For sequencing of the batches, they consider modifications of the ATC dispatching rule (Vepsalainen and Morton 1987). The results showed great computation time of these genetic algorithms.

Logendran et al (2007) presented six different search algorithms based on tabu search for minimizing weighted tardiness. A sequence-dependent unrelated parallel machine scheduling problem is investigated in this paper. Four different initial solution finding mechanisms, based on dispatching rules, are also developed in the hope of identifying better quality of initial solutions that might lead to identifying better quality final solutions. Suitable parameters of tabu algorithm to solve small, medium and large size problems follow from tests.

Colak and Keha (2008) solved the single machine total weighted tardiness problem by using integer programming and linear programming based heuristic algorithms. They discuss three methods (iterated optimization, stepped optimization-forward and stepped optimization-backward) to form the intervals and different post processing methods. Post processing methods are applied to the schedule found by ATC rule (Vepsalainen and Morton 1987). These algorithms significantly improve the solutions given by ATC heuristic.

1.1.3. Priority Rule

In following paragraph we discus rules for sequence determination of tasks in which they should be scheduled.

Vepsalainen and Morton (1987) published a study about priority rules for job shop problems with total weighted tardiness. This study considers scheduling of machine-constrained job shop with m machines and n tasks. Job i has m_i operations in a predetermined sequence on machines with deterministic processing times p_{ij} , $j = 1, \dots, m_i$. C_i is completion time of job i. There is a delay penalty, or weight, of w_i per unit time, charged if job i is completed after its due date d_i. This penalty, assumed to be constant over time, includes customer badwill, cost of lost sales or changed orders, and rush shipping cost. The objective is to minimize the weighted tardiness of the jobs: $\sum_{i=1}^{n} D_i$, where $D_i = \max\{C_i - d_i; 0\}$.

Rules in this study were in detail described and collated in tests. From these tests follows that the best rule for this problem is rule ATC (Apparent Tardiness Cost), which first time published Rachamadugu and Morton (1981). A little worse results attained rule weighted Convert (Carroll 1965), which is "predecessor" of ATC.

In the ATC heuristic, index $I_{ATC}(t)$ is calculated for every unscheduled job at time t as follows

$$I_{ATC}(t) = \frac{w_i}{p_i} exp\left(-\frac{max[(d_i - p_i - t); 0]}{k\bar{p}}\right)$$

where $\bar{p}$ is average processing time of all remaining unscheduled tasks, k is a look-ahead parameter, which is dependent on type of scheduling instance. This parameter is used to control the rate of discounting w_i/p_i and is in detail described in Rachamadugu and Morton (1981). The heuristic works as follows: we first identify the machine j that is available to process tasks at earliest (t denotes the time at which the machine is available). Next, we calculate $I_{ATC}(t)$ for all unscheduled tasks and schedule, at time t on machine j, the task that has the highest $I_{ATC}(t)$. The time t on the machine is updated and the procedure is repeated.

Lee et al. (1997) proposed an ATCS (Apparent Tardiness Cost with Setups) rule for a single machine when there are sequence-dependent setup times between the tasks. Setup time s_{li} is into effect when changing from job l to job i. Park et al. (2000) expanded ATCS rule about next parameter, which changes determination constants for priority equation. This change improved results of the objective function by average 6% over Lee et al. (1997) ATCS rule. Unfortunately, relation between new parameter and others constants is demonstrated only on neural network - numerical relation is not presented.

Gadkari, Pfund et al. (2007) expand ATCS rule by ready times (ATCSR rule). Pfund, Balasubramanian et al.(2007) applied ATCSR rule to a problem analogical to ours. Concretely, for scheduling semiconductor wafer fabrication facilities. The ATCSR index is given by

$$I_{ATCSR}(t, l) = \frac{w_i}{p_i} exp\left(-\frac{max[(d_i - p_i - \max[r_i, t]); 0]}{k_1\bar{p}}\right) exp\left(-\frac{s_{li}}{k_2\bar{s}}\right) exp\left(-\frac{max[(r_i - t); 0]}{k_3\bar{p}}\right)$$

where l is index of the last job completed on the machine which just has become free, $\bar{s}$ is the average of all setup time, r_i is the ready time of job i. Parameters k_1, k_2 and k_3 determine the relative importance of the exponential terms in relation to each other and WSPT term (w_i/p_i). These parameters are in detail described in study (Gadkari, Pfund et al. 2007).

Any of these rules leave out time-lags. To obtaining sequence of tasks in which the have be tasks scheduled, will be the best to use ATCSR rule.

1.2. Contribution

Two implemented scheduling algorithms are the result of this work. Efficiency of implementation of algorithm was investigated on these algorithms. General recommendations of implementation of scheduling algorithms were created from experience with implementation of these algorithms.

The first implemented algorithm is Iterative resources scheduling algorithm (Hanzálek and Šůcha 2009), which will be presented on multidisciplinary international scheduling conference[1] (MISTA 2009). I have cooperated to evolution of this algorithm. My task was to transform this algorithm to C++ and C# programming languages, to investigate efficiency of the implementation and to extend this algorithm to another objective function.

Filtered beam search algorithm (Schwindt and Trautmann 2003) was implemented to compare iterative resource scheduling algorithm. This algorithm was primarily modified to our problem and then it was implemented in Matlab and consecutively it was transformed to C#. The efficiency of the algorithm implementation was investigated during the implementation too.

The comparison of both algorithms shows their potentialities in application. Moreover, time symmetric mapping (Hanzálek and Šůcha 2009) was implemented inside the body of both algorithms. Thank this method we can obtain better results of these algorithms.

This work is organized as follows: Section 2 introduces definitions of basic terminology of scheduling area for this work. Three scheduling algorithms to minimizing $Cmax$ are presented in Section 3. An adaptation of the algorithm to objective function total weighted tardiness is described in Section 4. Section 5. presents general recommendations to efficient implementation of scheduling algorithms. Comparison of both implemented scheduling algorithms is showed in Section 6. and summary of all results is presented in Section 7.

[1] http://www.mistaconference.org

2. Problem Statement

2.1. Task

Set $T = \{T_1, T_2, \dots, T_n\}$ is set of n tasks. Each task in scheduling is characterized of several data (Blazewicz et al. 2001): *Processing time* p_j is the time needed process task T_j. *Ready time* (or arrival time) r_j is the time at which task T_j is ready for processing. *Due date* d_j specifies a time limit by which T_j should be completed. *Deadline* $\tilde{d}_j$ is a hard real time limit by which task T_j must be completed. *Weight* (priority) w_j expresses the relative urgency of T_j. *Start time* s_j specifies a time in which a task T_j started. *Completion time* C_j is the time when is T_j completed. Completion time we can expressed as: $C_j = s_j + p_j$.

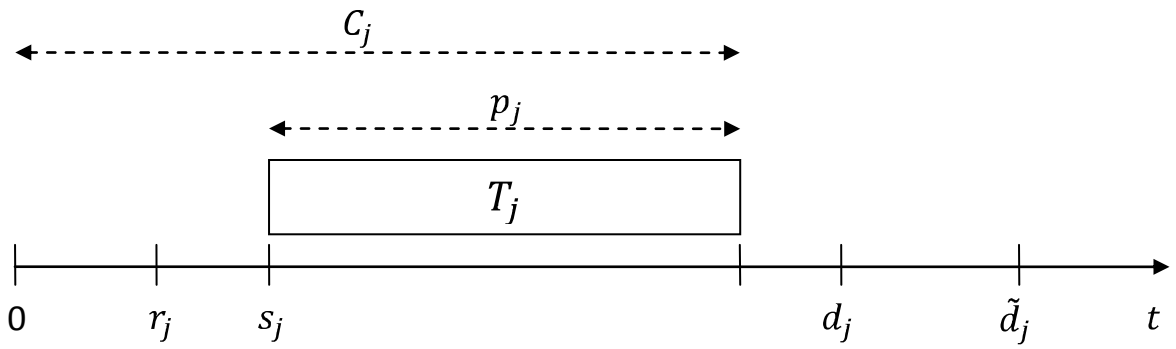


Fig. 1. Properties of task

2.2. Makespan (schedule length)

Makespan C_{max} is defined as a maximum from all completion time C_j.

$$C_{max} = \max(C_1, C_2, \dots, C_n)$$

2.3. Tardiness

Tardiness Dj of task j is defined as a difference between completion time and due date of task j, simultaneously Tardiness must be greater or equal than zero.

$$D_j = \max\{C_j - d_j; 0\}$$

Total tardiness is defined as $\sum_{j=1}^{n} D_j$, where n is number of all tasks.

2.4. Total Weighted Tardiness

Weighted tardiness D_w of task j is defined as tardiness of task j multiplied by a weight of task j.

$$D_w = D_j w_j$$

Similarly as tardiness we define total weighted tardiness as $\sum_{j=1}^{n} D_j\, w_j$.

2.5. Generalized Precedence Constraints (Time-lags)

A precedence relation $i \to j$ with meaning $s_i + p_i \leq s_j$ may be generalized (Brucker and Knust 2005) by a start-start relation of the form $s_i + d_{ij} \leq s_j$ with an arbitrary integer number $d_{ij} \in Z$. The interpretation of the relation $d_{ij} \in Z$ depends on the sign of d_{ij}:

If $d_{ij} \geq 0$, then task T_j cannot start before d_{ij} time units after the start of task T_i. This means that task T_j does not start before task T_i and d_{ij} is a minimal distance (time-lag) between both starting times.

If $d_{ij} < 0$, then the earliest start of T_j is $-d_{ij}$ time units before the start of T_i, i.e. T_i cannot start more than $-d_{ij}$ time units later than T_j. If $s_j \leq s_i$, this means that $|d_{ij}|$ is a maximal distance between both starting times.

If $d_{ij} > 0$ holds, the value is also called a positive time-lag or a minimal time-lag d_{ij}^{min}. If $d_{ij} < 0$, it is called a negative time-lag or a maximal time-lag d_{ij}^{max}.

Positive time-lag Negative time-lag

Fig. 2. Example of a positive time-lag and a negative time-lag

2.6. Setup Time

In a scheduling model with sequence-dependent setup times (or sequence dependent change over time) (Brucker and Knust 2005) the set of all tasks is partitioned into q disjoint sets $G_1, \dots, G_g$ called groups. Setup time o_{gh} is associated with each pair (g, h) of group indices.

For each $i \in G_g$ and $j \in G_h$, if i is processed before j, then the restriction $C_i + o_{gh} \leq S_j$ must be satisfied. For example, setup times may be used to model changeover times of a machine which occur when the machine is changed for the production of a different product (e.g. if a painting machine has to be prepared for a different color).

2.7. Processors

Dedicated processors (Blazewicz et al. 2001) are processors (or resources or machine) which can process only some type of tasks. *Parallel identical processors* can process all types of tasks and they work the same (identical) speed. If the speeds of the processors depend on the particular task processed, then they are called *unrelated processors*.

2.8. Multiprocessor tasks

Blazewicz et al. 2001 defined *Multiprocessor task* follows: We are given a set T of tasks of arbitrary processing times which are to be processed on a set $R = \{r_1, r_2, \dots r_m\}$ of m identical processors. There are also s additional types of resources, $A_1, \dots, A_s$, in the system, available in the amounts of $m_1, \dots, m_s \in N$ units. The task set T is partitioned into subsets,

$$T^j = \left\{T_1^j, \dots, T_{n_j}^j\right\}, j = 1,2, \dots k,$$

k being a fixed integer $\leq m$, denoting a set of task each requiring j processors and no additional resources, and

$$T^{jr} = \left\{T_1^{jr}, \dots, T_{n_j^r}^{jr}\right\}, j = 1,2, \dots k,$$

k being a fixed integer $\leq m$, denoting a set of task each requiring j processors simultaneously at most m_l units of resource type $R_l, l = 1, \dots, s$.

For example, in manufacturing environments materials, transport facilities, tools, etc. can be considered as additional resources.

2.9. Take-give Resources

Take-give resources (Hanzálek, Šůcha 2005) are needed from the beginning of a task to the completion of another task.

Set $Q = \{1, \dots, b\}$ is set of b take-give resources. *Take-give* resource $k \in Q$ has capacity of $Q_k \in Z^+$ units such that $Q_k < \infty$. Occupation i requires $a_{ilk} \in \{0,1\}$ units to take-give resource $k \in Q$ during its execution. Occupation i starts its execution at s_i, start time of task which takes a *take-give* resource, and finishes its execution at $C_l = s_l + p_l$, completion time of task which gives back the *take-give* resource.

2.10. Job-shop

The job shop problem is one of the classical scheduling problems. In the standard job shop scheduling problem (Bontridder 2005) a set of k jobs and a set of m machines are given. Each machine can handle at most one task at a time. Each job consists of a chain of n tasks. Each task has to be processed on a given machine during a time period of a given length. The purpose is to find a schedule such that the makespan is minimized.

2.11. Notation of Scheduling Algorithm

We use notation, proposed by Graham and Blazewicz (e.g. Blazewicz et al. 2001), for classification of scheduling problems. The notation is composed of three fields $\alpha|\beta|\gamma$. They have the following meaning: The first field α describes the processor environment, the second parameter β denotes task and resource characteristic and the third field γ describes an objective function (an optimality criterion).

2.12. Filtered Beam Search

Filtered Beam Search (see Neumann et al. 2003, Sect 2.5.) is one method from truncated branch-and-bound algorithms. This method is based on depth-first search. By α and $\beta < \alpha$ we denote the integers corresponding to the filter width and the beam width. After the generation of all child nodes of current node, we order them according to some filter criterion. The first α child nodes are evaluated on the basis of a beam criterion and the best β nodes are added to the enumeration tree. The remaining child nodes are excluded from further consideration.

For hard problem instances, even a beam width of $\beta = 2$ too large. For that reason we choose for each enumeration β from interval $[1,\ 2]$ randomly.

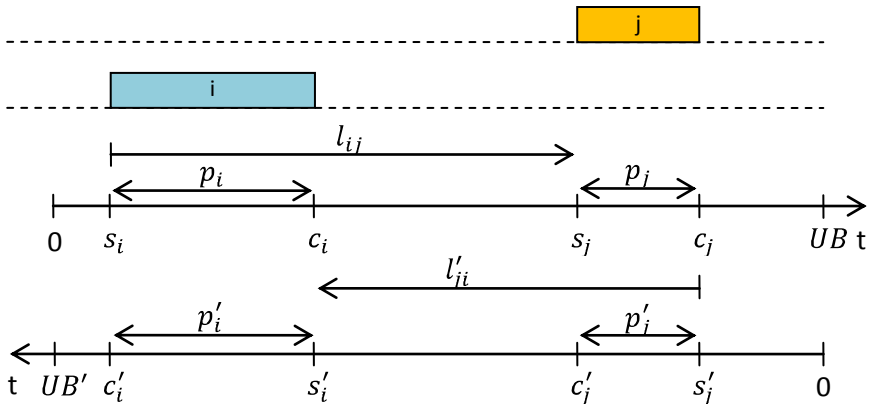
2.13. Time Symmetry Mapping

Time symmetry mapping (Hanzálek and Šůcha 2009) is a method which formulates how to construct a schedule in backward time orientation. A backward execution of a given schedule of $PS|temp, o_{ij}, tg|C_{max}$ problem is illustrated in this section. Basically there are two ways how to construct the schedule in backward oriented time while satisfying the temporal, resource and take-give resource constraints. The first way is to change the code of algorithm (re-implement a scheduling algorithm). The second way is to transform the input data and to run the original scheduling algorithm. The time symmetry mapping (TSM) deals with transformation of the input data.

Illustration of the TSM for properties of tasks and for take-give resources is shoved in Fig. 3. Definitions of TSM for instance of $PS|temp, o_{ij}, tg|C_{max}$ problem is following:

Take-give resources	$a'_{lik} = a_{ilk}$
The longest paths (between tasks)	$l'_{ji} = l_{ij} + p_j + p_i$
Changeover times	$o'_{ji} = o_{ij}$
Processing times	$p'_i = p_i$
Upper bound of instance	$UB' = UB$
Start times	$s'_i = UB - s_i - p_i$

TSM for tasks



TSM for take-give resources

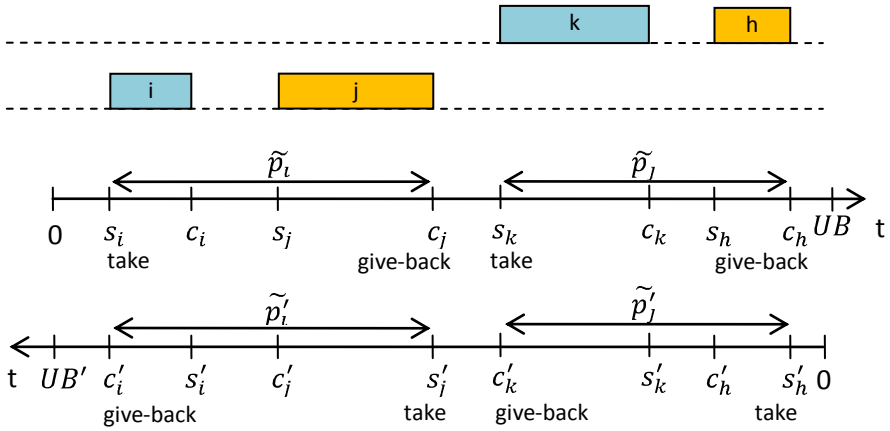


Fig. 3. Illustration of the time symmetric mapping for properties of tasks and for take-give resources.

2.14. Disjunctive Graph

The disjunctive graph $G = (V, C, D)$ is a graph (Brucker and Knust 2005, Blazewicz et al. 2001) with vertex set V, a set C of directed arcs (conjunctions) and a set D of undirected arcs (disjunctions). In connection with the job-shop problem G is defined as follows:

- The set V of vertices represent the set of all tasks. Two dummy tasks are added, they are representing the start and end of a schedule (tasks 0 and $n + 1$).
- The set C of conjunctions represents the precedence constraints between consecutive tasks of the same job. For every two consecutive tasks of the same job there is a directed arc. This arc is weighted with the processing time of the beginning task. The processing time of the dummy tasks are equal to zero.
- The set D of disjunctions represents the different orders in which tasks on the same machine may be scheduled. Each two tasks that require the same machine have disjunctive arc.
  - If we do not know order of tasks in schedule these arcs are non-oriented. This graph is called non-oriented disjunctive graph (see Fig. 4).
  - If we know order of tasks in schedule these arc are oriented. They are showed order in which are tasks execute on the same machine. This graph is called oriented disjunctive graph (see Fig. 5).

Example:

Instance of job-shop problem:

Job 1:

Number of task	1	2	3
Processing time	5	6	2
Machine	3	2	1

Job 2:

Number of task	4	5
Processing time	3	4
Machine	1	3

Job 3:

Number of task	6	7	8
Processing time	6	2	2
Machine	2	1	3

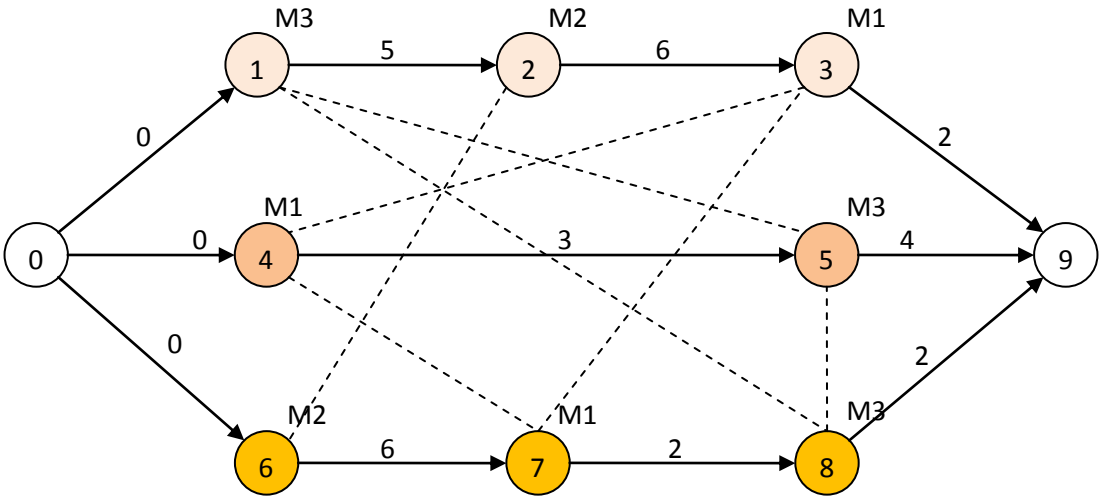


Fig. 4. The non-oriented disjunctive graph for instance job-shop above.

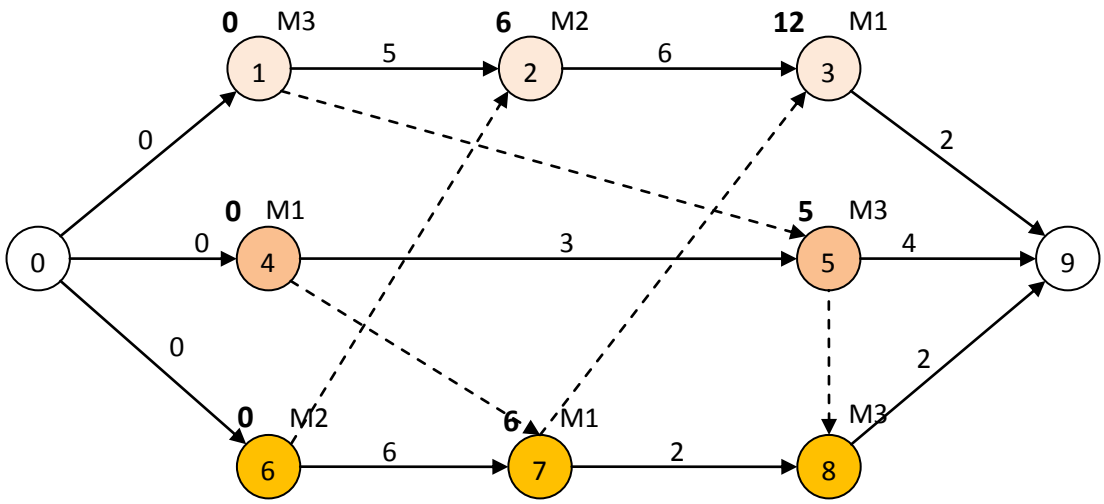


Fig. 5. The oriented disjunctive graph (one solution) for instance job-shop above. The bolt text represents starting times of tasks.

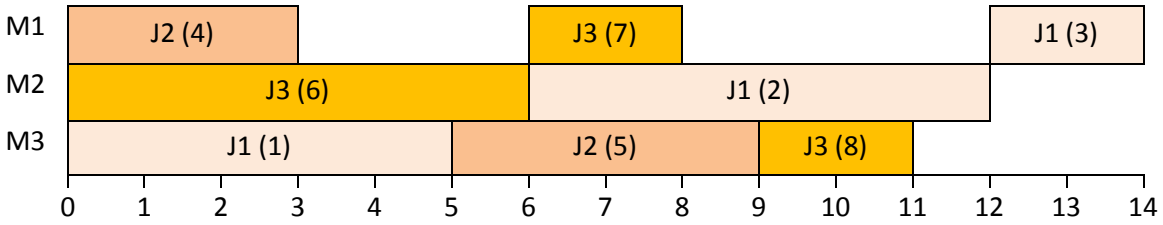


Fig. 6. The Gantt's diagram for Fig. 5. The oriented disjunctive graph (one solution) for instance job-shop above.

2.15. The Job Sequence on Machines:

The sequence on machine 1

Number of job	J2	J3	J1
Number of task	4	7	3

The sequence on machine 2

Number of job	J3	J1
Number of task	6	2

The sequence on machine 3

Number of job	J1	J2	J3
Number of task	1	5	8

2.16. Bierwirth's Sequence

Bierwirth (1995) introduces an alternative representation to job sequence on machines. This sequence is a sequence of job numbers which are ordered according to theirs start time. The Bierwirth's sequence of oriented disjunctive graph above is: J2 J3 J1 J2 J3 J1 J3 J1. This sequence is called the sequence with repetition too. Bierwirth's sequences can be efficiently generated by any greedy algorithm or any iterative method. These sequences can be used as chromosomes in genetic algorithm.

3. Algorithms For Minimizing $Cmax$

Three algorithms for minimizing objective function $Cmax$ are described in this chapter. The first one, Iterative Resource Scheduling (IRS) algorithm (Hanzálek and Šůcha 2009) is a priority-rule based method with unscheduling step where tasks are scheduled successively according to the given priority rule. An efficiency, implementation and testing of this algorithm is main part of this thesis.

Second one, Filtered Beam Search (FBS) algorithm (Schwindt and Trautmann 2003) is based on a branch and bound method - Filtered Beam Search method. The last algorithm, Memetic algorithm (Caumond et al. 2007) is based on genetic algorithm and a powerful local search procedure. After studying of these algorithms it was decided that FBS algorithm will be implemented and compared with IRS algorithm. Description of memetic algorithm is retained for illustration of other possibilities for minimizing objective function $Cmax$.

3.1. Iterative Resource Scheduling Algorithm

Heuristic algorithm for project scheduling with time windows and take-give resources was published by Hanzálek and Šůcha (2009). The problem that is addressed is motivated by a real scheduling problem i.e. a lacquer production (Behrmann et al. 2005). They extend classical resource constrained project scheduling by a take-given resources (see Section 2.9). This problem which solve IRS algorithm can be denoted by $PS|temp, o_{ij}, tg|Cmax$. Moreover, they discussed how to construct a schedule in backward time orientation and they define as the time symmetry mapping (see Section 2.13). Heuristic algorithm IRS for the problem with take-give resources is described in the following section.

3.1.1. IRS Algorithm

The iterative resource scheduling algorithm (IRS), based on the iterative modulo scheduling algorithm (Rau 2000), is described in this section. The meaning of interval bisection method used in this algorithm is described in the first paragraph. The function $foundSchedule$ which tries to find a feasible schedule is described in the second paragraph.

Algorithm IRS tries to find a feasible schedule with schedule $Cmax \leq C$. The schedule length C is determined by interval bisection method from interval $C \in \langle LB, UB \rangle$. An upper bound UB denotes an upper bound of schedule length (see Brucker 1999). Similarly, LB denotes a lower bound of

schedule length. If the feasible schedule S is found by the function $foundSchedule$, the tasks are shifted to the left side in function *shiftLeft(S)* and upper bound is decreased $UB = Cmax(S_{left}) - 1$. Contrariwise, if the feasible schedule is not found, then the lower bound is updated as follows $LB = C + 1$.

```
IRS (Instance, BudgetRatio)
    Budget = BudgetRatio * n
    Calculate LB, UB
    C = LB
    Calculate priority                    /* longest path from T_i to T_{n+1} */

    While LB ≤ UB do
        S = findSchedule(C, priority, budget)
        If S is feasible THEN
            S_left = shiftLeft(S)
            UB = Cmax(S_left) − 1
            S_best = S_left
        Else
            LB = C + 1
        End
        C = (LB + UB/2)
    End
End

findSchedule (C, priority, budget)
    S = { }

    While Budget > 0 & |S| < n + 2 do
        i = arg max (priority)
        calculate ES_i, LS_i
        si = findTimeSlot (i, ES_i, LS_i)
        S = scheduleTask (I, si , S)
        /* Rotate of instance - TSM */
        Budget = Budget − 1
    End
    Return S
End
```

Function *foundSchedule* tries to found a feasible schedule in $Budget = n * BudgetRatio$ scheduling steps. The parameter $BudgetRatio$ is an input parameter of the algorithm and it is the ratio of maximum number of activity scheduling steps to the number of tasks n. This parameter is usually equal to 2 (see Section 6.2.1). Function $foundSchedule$ constructs a schedule according to the priority of tasks (vector $priority$). Priority of task T_i is given by the longest path from task T_i to the latest task in the schedule T_{n+1}(so called dummy task, see definition of RCPSP/max in Section 1). Task with highest priority, which was not scheduled yet, is chosen for scheduling. The earliest and the latest start time of the task, ES_i and LS_i respectively, are calculated for this task. Function $findTimeSlot$

finds the first $s_i \in \langle ES_i, LS_i \rangle$ such that there are no conflicts on the resources. If there is no such s_i then s_i is determined according to whether task T_i was scheduled once. If the task is being scheduled for the first time, then $s_i = ES_i$ otherwise $s_i = s_i^{prev} + 1$ where s_i^{prev} is the previous start time of task T_i. Function $scheduleTask$ schedule the task at s_i. Conflicting tasks, with respect to temporal or resource constraints, are unscheduled.

3.1.2. Time Symmetry Mapping Inside Algorithm

TSM (see Section 2.13) is method which defines how to construct a schedule in backward time orientation. This method is used in two ways in this work. An instance is reversed before start of algorithm and this reversed instance is input parameter of algorithm. Thus IRS algorithm is not modified. Second way is to use TSM inside of IRS algorithm.

Reversion of schedule inside IRS algorithm works at follows. We execute the algorithm several scheduling iterations with the original (forward) problem and then we use TSM and the same number of scheduling steps with backward time orientated instance. IRS algorithm created the schedule according to a vector of priorities. This vector must be calculated for both orientation of the problem. Thus we have two vectors of priorities, one for original problem and one for problem obtained by TSM. The final schedule is scheduled "from both sides" (several scheduling steps from the front and several scheduling steps from behind).

This method can "release" some tasks and then IRS algorithm can resolved some complicated instances. The best number of scheduling steps after which is suitable to reverse the instance was not found out. The instance was reversed after 16, 32, 64 scheduling steps.

3.2. Filtered beam search algorithm

Algorithm published in Schwindt and Trautmann (2003) (Scheduling the production of rolling ingots: industrial context, model, and solution method) was chosen to benchmark algorithm IRS. Solution is based on the branch-and-bound algorithm, concretely filtered beam search method (see Section 2.12). They created a scheduling algorithm for a real scheduling problem, rolling ingots production. Rolling ingots are starting material for the rolling of sheet, foil and strip, which are mainly used in the automotive, packaging, printing and construction industries. This algorithm is dedicated to solve this manufacturing problem only. In this section we describe the algorithm and then we discuss an adaptation of the algorithm to a more general scheduling problem.

3.2.1. Introduction and Basic Concept of Model

The production flow is showed in Fig. 7. In a potroom (a melting furnace), the ingredients composing the alloy are smelted in an elecrolytical process. Several alternative potrooms are available. Several casting unit belong to each potroom. Each casting unit is created of mold and stool-cap. The mold determines the cross-section of the ingot. Stoll-cap closes the bottom of the mold at the start of the casting process. Casting units are separated basely maximum cast length, which implies that not every ingot cannot be produced on each casting units. Casting system has casting units with the same length only. After the casting process, the ingot stays in the casting unit for cooling for a while. All ingots produced within one casting are of the same alloy and same length. The casting has to be started and completed at the same time for all casting units of a casting system. When an ingot with a different cross-section is performed, the mold of the casting unit has to be changed. The changeover can be performed only when any casting is in process.

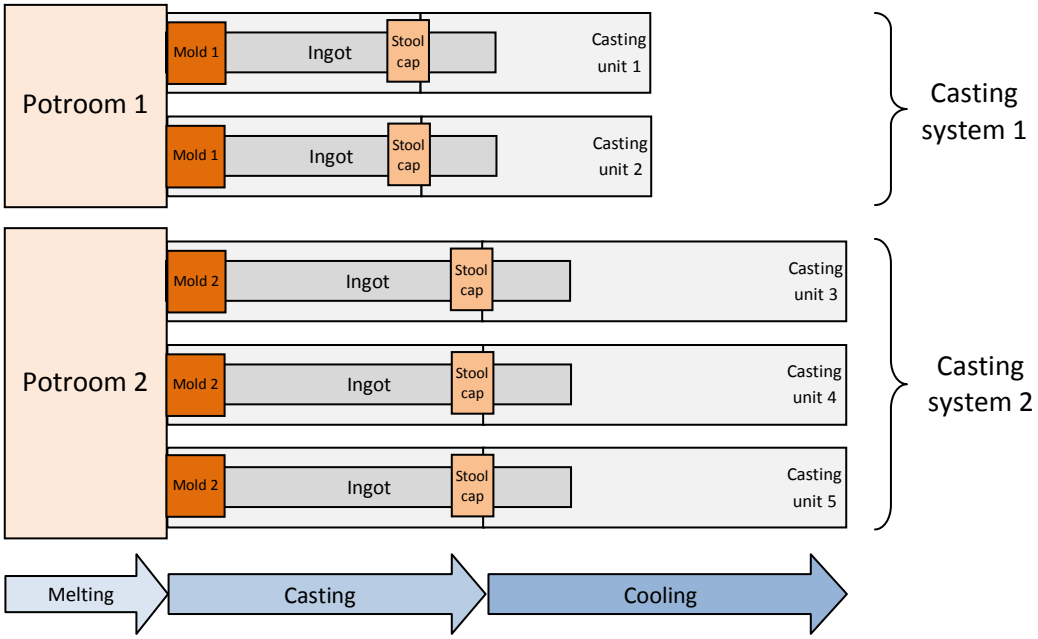


Fig. 7. Rolling ingots product flow.

Production order for ingots is characterized by their alloy and length, the production scheduling problem consists of computing a feasible production schedule with minimal makespan. In scheduling terminology we can describe production of individual ingots as jobs. Each job consists of the three operations (tasks). Task T_1 corresponds to melting in potroom, T_2 corresponds to changeover of the mold and T_3 corresponds to casting plus cooling in casting unit. Each task has different processing time and uses different resources. This is showed in Fig. 8. The job is described as task-on-node network. The arcs correspond to time lags between tasks. The jobs may be performed in alternative casting systems, for that reason we define modes. Then mode describe on which casting system is a job executed.

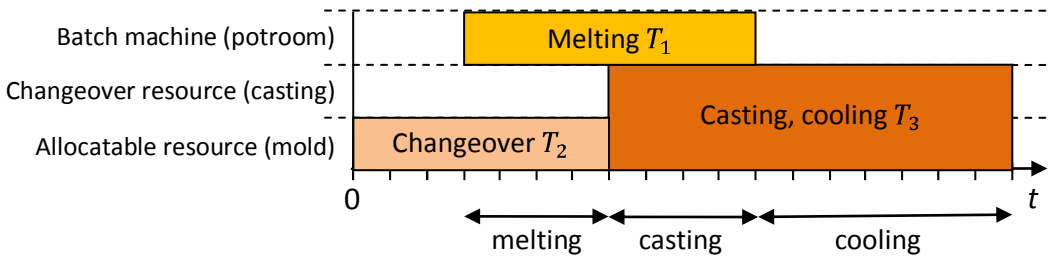


Fig. 8. Operations of job

3.2.2. Solution Method

Feasible solution must satisfy following constraints:

- The necessity to select a mode for each job
- The limited capacity of the renewable resources
- The requirement that operations running in parallel on batching machines must be of the same batching type and must be started jointly
- The need for changeover operations with sequence-dependent durations.

If we relax all those constraints, the remaining problem consists of scheduling all tasks subject to the temporal constraints. This temporal scheduling problem represents a longest path problem in task-on-node network. The solution may be infeasible if there are jobs for which no mode has been selected so far or one of the resource constraints may not be met. In first case we assign a mode to some job whose mode has not been fixed yet. The second case is resolved by introducing suitable time lags to task-on-node network.

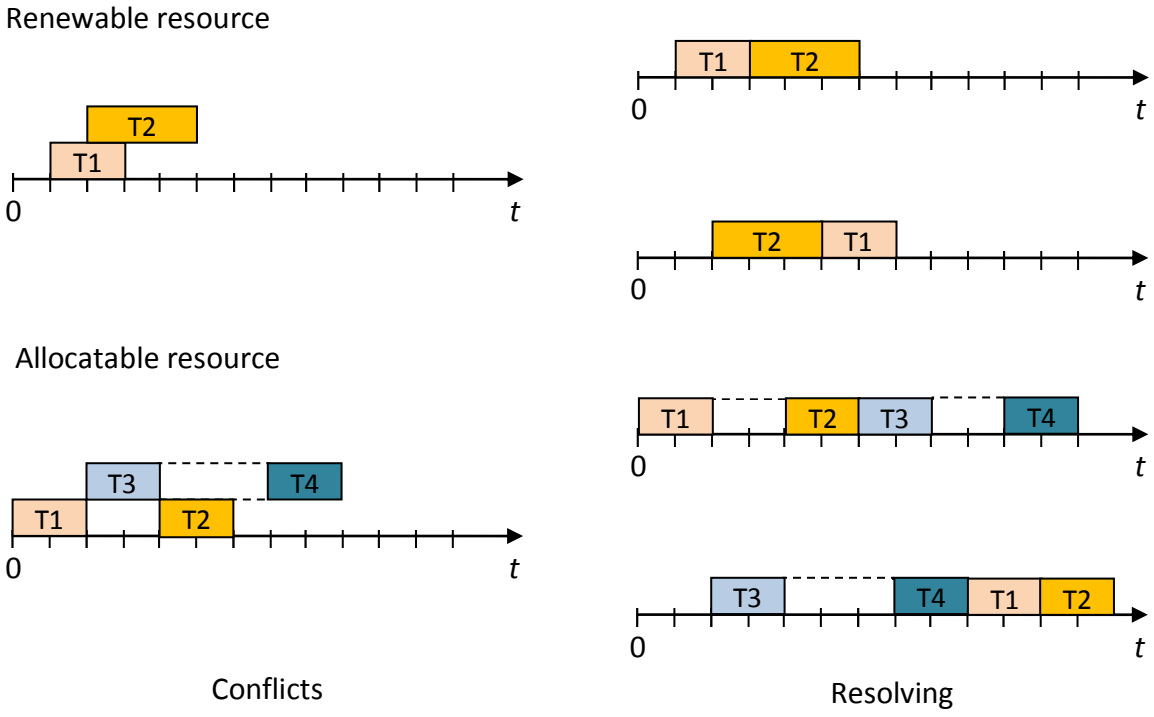


Fig. 9. Resolving capacity and allocation conflicts.

Resource capacity and allocation conflicts resolving is showed in Fig. 9. Capacity conflict may be resolved when shift the start of task T_2 up to the completion of task T_1 (or reverse) by introducing the positive time lag $D_{12}^{min} = p_1$ (or $D_{21}^{min} = p_2$). Allocation conflicts may be resolved similarly. In this case we shift allocation task T_3 behing operation T_2 by introducing positive time lag $D_{23}^{min} = p_2$ (or reverse). Changeover conflicts can be removed by new positive time lag $D_{12}^{min} = p_1 + s_{12}$ or reverse $D_{21}^{min} = p_2 + s_{21}$ (see Fig. 10.).

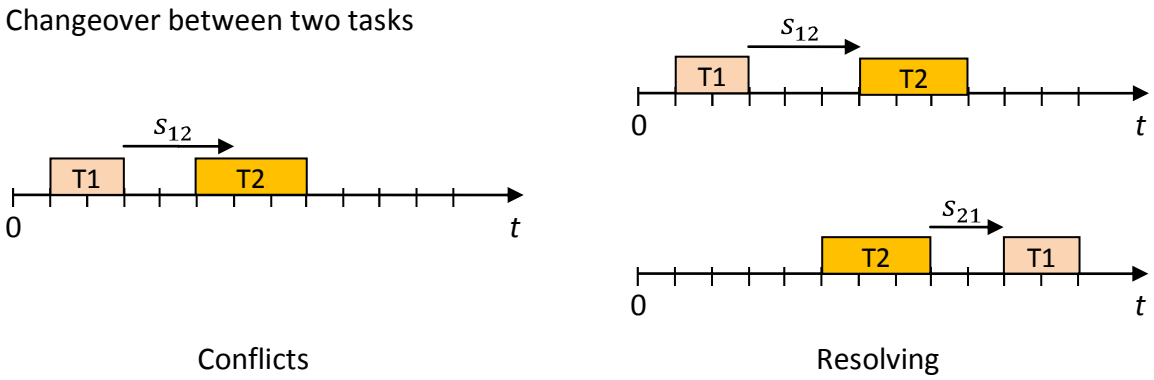


Fig. 10. Resolving changeover conflicts.

Batching machine conflicts are shoved in Fig. 11. The first case shoves two tasks which belong to different batching types ($b_1 \neq b_2$) but require the same batching machine. This problem can be resolved similarly as capacity conflict. In second case shoves two task which belong to the same batching types ($b_1 \neq b_2$) but start time T_1 is smaller then T_2 ($s_1 < s_2$). When are two tasks which belong to the same batching types, than those task must start together or must start in another batch.

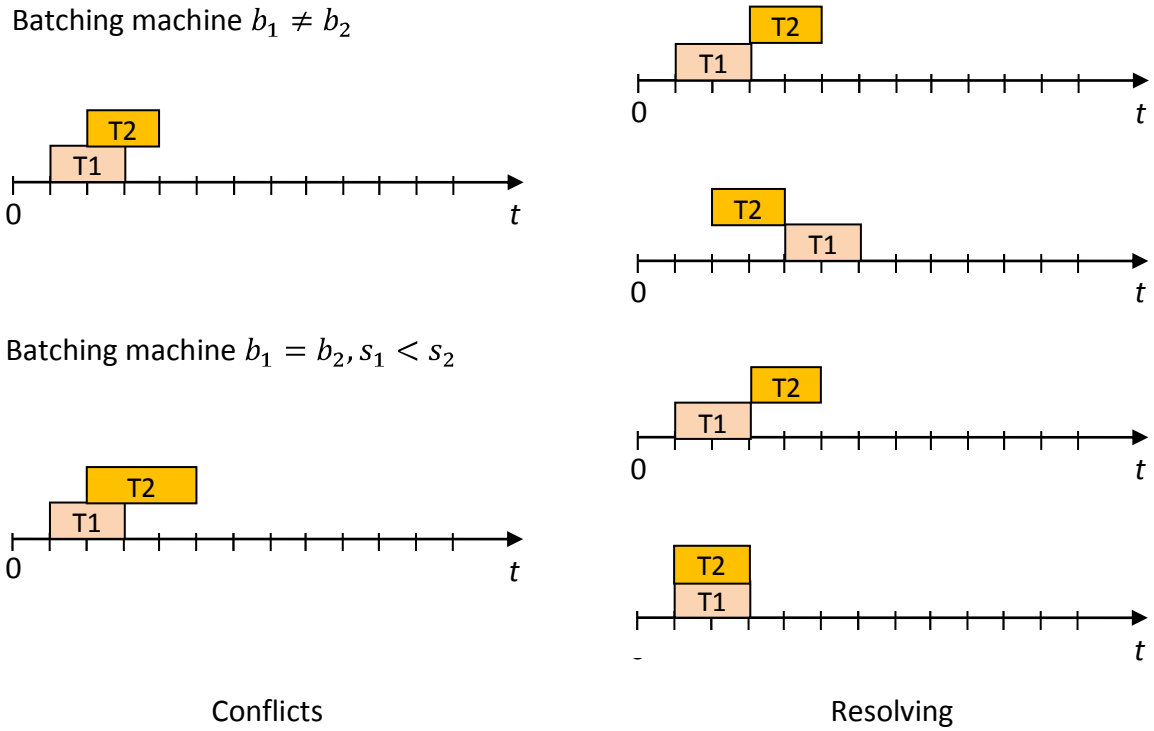


Fig. 11. Resolving batching conflicts.

Now can be described the filtered beam search procedure (see Section 2.2.12) for this problem. The algorithm first enumerates number of conflicts for each task. Tasks are sorted by number of conflicts. Tasks which have batch conflict are favored and tasks which have changeover conflict are penalized. Subsequently, is chosen α tasks with minimal number of conflicts. For those α best cases we resolve conflicts, find the longest paths and calculate the Total Displacement. Total Displacement TD = $\sum_{i=1}^{n}(S_i' - S_i)$, where n is number of tasks, S is a vector of start times tasks source node, S' is vector start times tasks new (resolved) node. Filter width α is equal 3 for all instances. For β nodes with minimal total displacement is procedure repeated. Even a beam width of $\beta = 2$ is too large for hard problem instances. Let $\beta = [\bar{\beta}u]$ denote the integer random variable where u is uniformly distributed in interval $0 < u \leq 1$ and $\bar{\beta} = 2$ for $n \leq 10$, $\bar{\beta} = 1.8$ for $10 < n \leq 30$ and $\bar{\beta} = 1.6$ for $n > 30$. Search continues until feasible solution is found. Pseudocode of the algorithm is shown below:

```
Main (Instance, Alpha, Beta)
        S = the longest path between all tasks and source dummy task
        x = assign modes

        FBS (S, x)
End

/* Recursive procedure */
FBS (S, x):
        allConflicts = Find conflicts between tasks      /* - Make favorable for tasks which have
                                                          *   batch conflict
                                                          * - Make handicap for tasks which have
                                                          *   changeover conflict */
        If allConflicts is empty
                If Cmax(S) < CmaxBest
                        Cmax best = Cmax(S)
                        Start time best = S
                        Modes best = x
                End
                Return
        End

        conflicts = choose Alpha conflicts from allConflicts with minimal number of conflicts

        For each C from Conflicts do
                R (S', x') = Resolve conflict (C)  /* add positive time lag or assign mode and
                                                      * find the longest path for all tasks */
                TD = Total Displacement between S' and S
        End do

        R = Choose Beta resolved case with minimal Total Displacement from R
        For each r from R do
                /* Rotate of instance - TSM */
                FBS (S', x')
        End do
End
```

3.2.3. Adaptation of Algorithm to Our Problem

Our problem is different from problem described in Schwindt and Trautmann (2003). However, we can adapt this algorithm easily. Now we will describe differences between both algorithms. Our problem uses neither batch machine nor modes. For that reason, we can omit these constraints.

In reverse, multiprocessor tasks is not considered in Schwindt and Trautmann (2003). It is not problem because searching of multiprocessor conflicts may be performed separately like searching of one processor conflicts. Then we must verify all processors which belong to one task. For example, we have a task which must be performed on three processors. Then we must check that are not any conflict on any of the three processors.

Changeover time between two is applied in case both tasks are performing on the same processor only. Thus we must keep information about processors assignment. If we find two tasks on the same processor at the same time, we have two possibilities to resolve it. We can assign one task on another (free) processor or we can add positive time lags between tasks (in the same way like in paper). We assign all tasks on the first processor at the beginning of the algorithm. Note if we would propose suitable processors assignment at the beginning of the algorithm, can we have less conflicts at the beginning of the algorithm.

Conflicts on batch machine are advantage and conflicts of changeover time are disadvantage in original algorithm. We do this similarly. Take-give resources conflicts are resolved at first and conflicts with changeover times are resolved at last.

Incremental Algorithm (distance matrix D, arc from i to j with weight wij)

If $w_{ij} > -D_{ji}$
    Return $D = 0$ // Arc produces cycles with positive length
End

For each g, h from D
    If $D_{gh} < D_{gi} + w_{ij} + D_{jh}$
        $D_{gh} = D_{gi} + w_{ij} + D_{jh}$
    End
End

Return D // Return updated distance matrix

There are many algorithms for enumeration longest path in task-on-node network. We chosen an incremental algorithm presented by Bartusch et al. (1988). This algorithm (see above) update distance matrix D when adding some arc w_{ij} (from node i to node j with weight w) to the network. Moreover, it finds out if adding arc w_{ij} produce cycle of positive length or not. Time complexity of this algorithm is $\mathcal{O}(n^2)$, where n is number of nodes. We must compute distance matrix from task-on-node network at the beginning of algorithm. For this problem we use well known Floyd-Warshall algorithm.

3.2.4. Time Symmetric Mapping Inside Algorithm

The TSM (see Section 2.13) inside FBS algorithm is used in the similarly way as in the IRS algorithm (see Section 3.1.2). In this case the problem is not reversed after several scheduling steps, but it is reversed after several resolved conflicts. The instance is reversed before creating new children in solution tree. For backward orientation of instance the different conflicts are found and hence the algorithm can found different solution of problem. From our tests follow, that the instance is advantageous reversed after 10, 20, 50 resolved conflicts.

3.3. Memetic Algorithm for the Job-shop with Time-lags

Algorithm published by Caumond et al. 2007 (A memetic algorithm for the job-shop with time-lags) was chosen to benchmark algorithm IRS. This paper addresses the job-shop scheduling problem with minimal and maximal time-lags (see Section 2). The algorithm is based on a memetic algorithm and a powerful local search procedure. The problem is modelized as a non-oriented disjunctive graph (see Section 2). Since a job sequence on machines is generated, it is possible to obtain an oriented disjunctive graph. A Bellman like longest path algorithm permits to compute the earliest completion time of the last operation: the makespan. The makespan denotes the completion time of the last operation. Individual parts of this algorithm are described in next paragraphs.

A solution is an oriented disjunctive graph. The arcs between operations of jobs which use the same machines define the operations sequence on machines. Bierwirth sequence (Bierwirth 1995) is used in algorithm as alternative representation of job sequence on machines. Bierwirth's sequences represent chromosomes in genetic search process.

Unfortunately there exist inconsistent oriented disjunctive graphs which contain positive cycle length. The positive cycles in the graph are due to incorrect orientation of edges in arc but also due to negative arcs in the graph. When no positive cycle exists in the graph, a Bellman like longest path algorithm permits to determine the start time of each task.

Positive cycle can be efficiently detected during the longest path algorithm run. Thanks to Bierwirth's sequence, cycles positive length are only due to maximal time-lags and not to incorrect operation sequence on machines (incorrect disjunctions between operations on one machine are not possible). The main problem is to determine which maximal time-lags must be removed from the graph to obtain a graph without positive cycles. This problem solve following algorithm:

Procedure Evaluate_A_Sequence:
Evaluate the Bierwirth's sequence without maximal time-lags in graph.
$f_1(x) = 0$
For i = 1 **to** N **do**
    Checked the maximal time-lags of operation i
    Evaluate the Bierwirth's sequence with the maximal time-lags
    which have been checked

$f_2(x)$ = Completion time of the last operation
if (there is positive cycle in the graph) **then**
$f_1(x) = f_1(x) + 1$
Unchecked the maximal time-lag of operation i
End if
End do

In algorithm N is number of operations in Bierwirth's sequence x, $f_1(x)$ is the number of time-lags which must be unchecked (removed from graph) to obtain a graph without cycle and $f_2(x)$ is completion time of a Bierwirth's sequence.

The function $f(x) = w_1 f_1(x) + w_2 f_2(x)$ defines a cost $f(x)$ for each Bierwirth's sequence x. If w_1 and w_2 are chosen such $Min(w_1 f_1(x)) > Max(w_2 f_2(x))$, $f(x)$ is strict hierarchic function which affects a value to Bierwirth's sequence with the following properties:

- A Bierwirth's sequence without unchecked maximal time-lags has a lower cost than any Bierwirth's sequence with unchecked maxima time-lags.
- if two Bierwirth's sequence have unchecked maximal time-lags, then the Bierwirth's sequence with the lowest number of unchecked maximal time-lags has the lowest cost.

The procedure Evaluate_A_Sequence is a greedy procedure because it unchecks the last checked maximal time-lags when a cycle is detected.

How was it already said, the Bierwirth's sequences are considered as a chromosomes. These chromosomes have two properties: chromosomes can be efficiently evaluated with respect to their sequence; two chromosomes can hold to the same oriented disjunctive graph. The fitness of the chromosome is evaluated by function $f(x)$ which is described above. Feasible chromosome is chromosome with $f_1 = 0$ (graph with no cycle).

Initial population is created by priority dispatching rules generation (a heuristic) and by random chromosome generation. The main reason to combine heuristic chromosomes and random chromosomes in the initial population is to obtain a great diversity. The heuristic generation is based on the framework folly described in Caumond et al. 2005.

The crossover is based on the GOX crossover first introduced by Bierwirth 1995. The main characteristic of this crossover is to preserve the relative order of tasks. Note, that a child obtained after crossover of two feasible parents, may not be feasible.

The mutation is here realized by local search. Local search is based on an exchange of two tasks in machine-block. A machine-block is a sequence of tasks of the same job processed consecutively. We consider these sequences as sequences of machine-block on a critical path (the longest path). The better solution can be obtained by exchanging one task at the end of one block with another one at the beginning of the next block. This new sequence is transformed into a chromosome and a new cost evaluated.

Memetic algorithm has few parameters:

mni	maximal number of iterations
np	maximal number of unproductive iteration before a restart
nc	number of chromosomes in population
pm	local search probability
nm	maximal number of iterations during local search
pr	percent of population

The Algorithm selects two chromosomes to undergo crossover and mutation. The resulting child replaces one existing chromosome in population. The GOX crossover is applied to two chromosomes and one child is selected at random. The child undergoes a local search with probability *pm* (local search probability). If is not the fitness of the child the duplicate, the child will be mutated and productive iteration counted. If is the fitness of the child the duplicate, the child will be not mutated and this iteration is an unproductive one. When the maximal number of unproductive iterations is reached (*np*), the algorithm experienced a restart by replacing *pr* percent of population (*pr*) by random generated chromosomes. Note that the best chromosome cannot be mutated during a restart and so the best chromosome is preserved. The memetic algorithm stop when the maximal number of iterations (*mni*) is reached or the lower bound is reached. The block scheme of the algorithm is in Fig. 12.

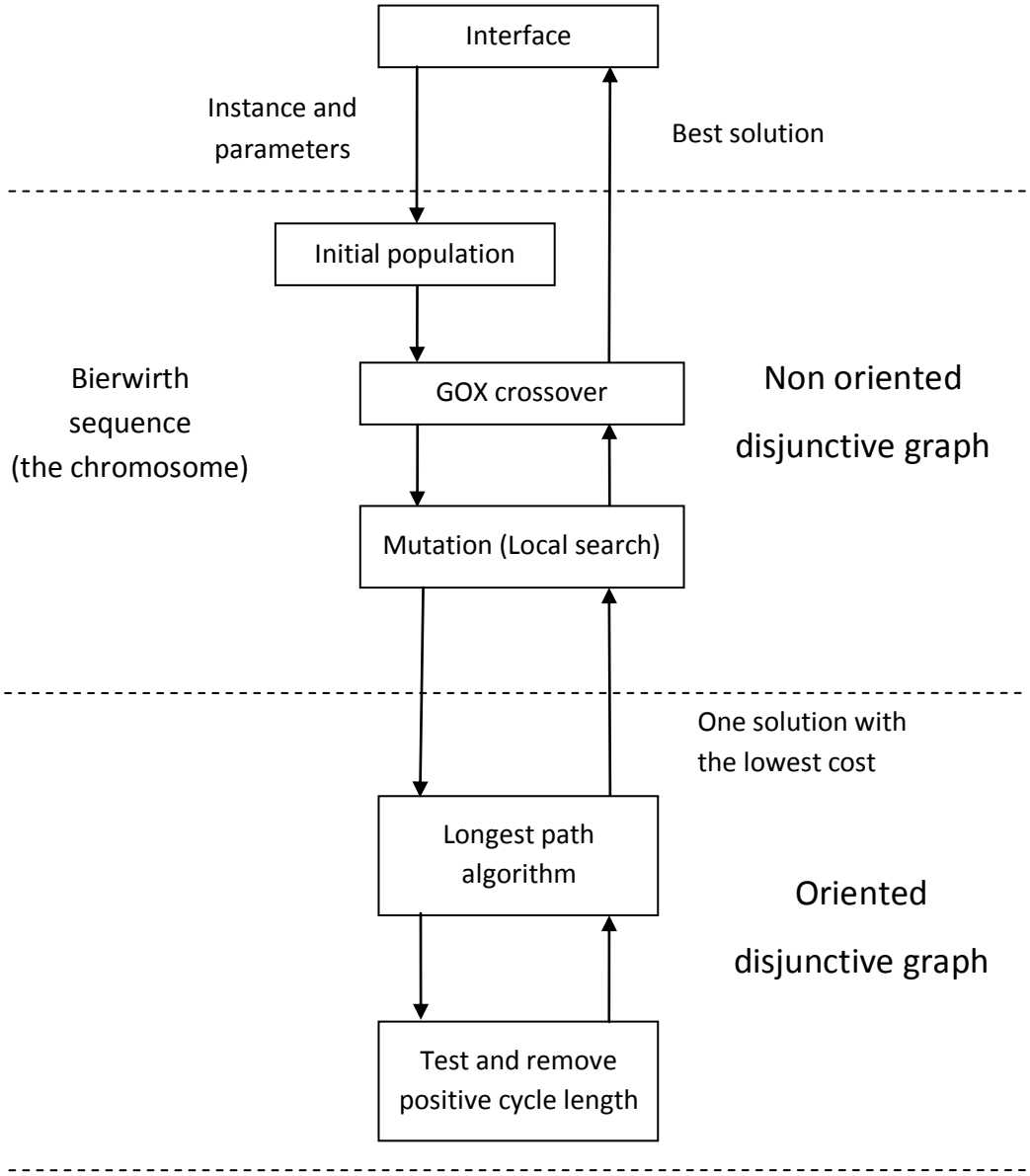


Fig. 12. The block scheme of alternative algorithm.

4. An Algorithm For Minimizing $\sum w_j D_j$

An algorithm for minimizing Total Weighted Tardiness (definition is in Section 2.4.) is described in this section. Basic idea was modified IRS algorithm (Hanzálek, Šůcha 2009) for minimizing this function. IRS algorithm stayed the same out of following parts. I. Priority of tasks (vector priority) is counted with the ATCSR rule. This priority rule for minimizing TWT is described in detail in Section 1.1.3. II. Modified algorithm not iterate over interval (LB, UB). A new schedule (output parameter from *foundSchedule* function) is analyzed and parameters of instance or parameters of ATCSR rule are changed in the next iterates.

Some methods were suggested for solution of this problem and their results were compared with optimal schedules. Optimal schedules were found out by ILP. Independent versions of algorithm were created for each of methods. The algorithm was created for $PS|temp|Cmax$ problem for simplicity. For this reason, the ATC rule is sufficient

$$I_{ATC}(t) = \frac{w_i}{p_i} exp\left(-\frac{max[(d_i - p_i - t); 0]}{k\bar{p}}\right)$$

Methods are described in detail in the followings paragraphs. Summarize of results all methods is presented in the end of this chapter.

4.1. Basic Concept (TWT_0)

ATC Rule determines the priorities of tasks for scheduling so that the objective function TWT is minimizing. The calculation of priorities is always started before selection of task which should be scheduled. Thus, the calculation of ATC indexes is started in loop function *foundSchedule* in IRS algorithm. All parameters for calculation of ATC index are parameters of task. The constant k is calculated from parameters of whole instance. This constant is described in Park, Kim, Lee (2000) in detail.

The input instance must include set of due dates and weights for all tasks. It is important to right calculate of ATC rule, but this is not guarantee in all cases. For this reason, these instances must be adapted. This action is introduced as propagation of due dates and weights between all tasks and is described in next subsection. All successors version of algorithm include this basic concept.

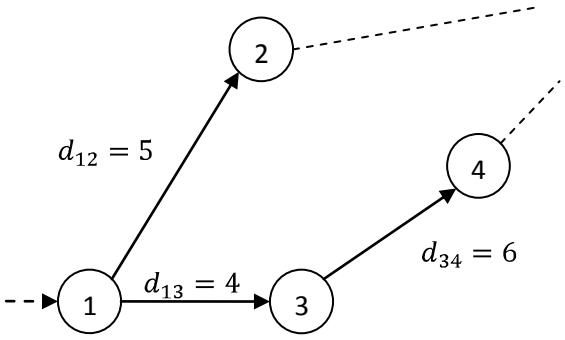
4.1.1. Propagation of Due Dates and Weights Between All Tasks

Propagation set due dates and weights of the tasks which have not this values set. Due date d_i for task T_i is counted as follows

$$d_i = \min_{\forall j:\, l_{ij} \neq -\infty,\ d_j \neq \infty} (d_j - p_j - l_{ij} + p_i)$$

where l_{ij} is the longest path from task T_i to task T_j. If $l_{ij} = \infty$ then do not exist any path from T_i to T_j. If $d_j = \infty$ then the due date for this task is not set.

Weight w_i of task T_i is counted as mean weight of tasks, which are successors of task T_i (if the path l_{ij} exist). Propagation of due dates and weights is shoved in Fig. 13.



Input set of tasks

i	1	2	3	4
p_i	2	5	4	3
d_i	∞	9	10	15
w_i	0	10	5	5

New set of tasks

i	1	2	3	4
p_i	2	5	4	3
d_i	1	9	10	15
w_i	7	10	5	5

Fig. 13. Propagation of due date d_1 and weight w_1.

4.2. Shift Left of Earliest Start (TWT_1)

Analysis of schedule and iteration are introduced in this version of algorithm. Schedule found by function *foundSchedule* is evaluated by criterial function ($\sum D_j w_j$). The task with the biggest penalty may be found. Earliest start of this task is decreased by next entrance to *foundSchedule* function. These actions correspond to one iteration of algorithm. If feasible schedule is found then the biggest penalize task will be decreased again. New earliest start $newES_i$ for tasks T_i is counted as follows

$$newES_i = s_i - round((s_i + p_i - d_i)/nIter)$$

the variable $nIter$ describe number of unexecuted iterations. It means, the more is executed of iterations ($nIter$ is decreased) the more is decreased new earliest start of the biggest penalize task. Our tests showed that 8 iteration of algorithm is sufficient. Comparison with optimal solutions is showed at the end of this chapter.

4.3. Increase Weight/Decrease Due Date of Tasks

This method is based on increase of weight and decrease of due date for tasks which are the most penalize. This modification of properties of tasks produces the bigger priority for these tasks. This method can be effective, but estimate of these changes is not easy. If the change of these properties is small then priorities will be the same. If the change of these properties is too big then the algorithm will go to invalid way. Border between these extremes was not found. The results of this method were wrong and therefore they are not shoved in the comparison.

4.4. Jump over of Tasks (TWT_2)

Tasks which have the biggest penalize are scheduled before the rest of the tasks in this method. Concretely, it is allowed for these tasks be scheduled although they have smaller priorities. Order tasks for scheduling is different than in basic concept-exactly according to ATC rule. Because of it, a total penalize can be smaller. Number of jumped tasks increase for the biggest penalize task. This number represents how much of task with the higher priorities can be jumped. This method is presented on a simple example (Table 1).

Table 1
Comparsion of order tasks for scheduling obtain by jump task method.

Task i	1	2	3	4	5
I_{ATC}	2,54	0.65	8,44	0,23	15,3
Order tasks for scheduling	**3**	**2**	**4**	1	5
Jump over of tasks	0	0	2	0	0
New order tasks for scheduling	**4**	**3**	**2**	1	5

4.5. Iteration over Constant k (TWT_3)

This method is inspired by article Park, Kim et al. (2000). Constant k is not counted from parameters of instance, but constant is chosen from the recommend interval. The algorithm is running several times, each time with different constant from recommend interval. It is simple to extend our algorithm by this method. This constant may be changed by the calling method *foundSchedule.* This method shows that counting constant k from instance is not a robust solution.

4.6. Combination of Methods (TWT_4)

The last method unites two best methods only, *Iteration over Constant* k and *Jump Tasks*. The TWT_3 is started at the beginning. During its execution the best constant k is found. Then TWT_2 method (with found constant k) is started.

4.7. Results

All methods were tested on the same instances and compared with the optimal solution (found by ILP). Thus, comparison is accurate. These results are summarized in Table 2. We can see that difference objective function between IRS algorithm and optimal solution is smallest for methods *Iteration over Constant* k (TWT_3) and its modification (TWT_4).

Table 2
Comparison of IRS algorithms and ILP for Total Weighted Tardiness objective function.

	Difference objective function between IRS and ILP [%]				
n **tasks [-]**	**TWT_0**	**TWT_1**	**TWT_2**	**TWT_3**	**TWT_4**
5	12,9	11,9	4	2,5	1,3
10	23,6	20	16,6	11,8	9,7
15	33,9	28,9	25,1	16,9	14,5

But it is obvious that all methods have a common problem. For increased number of task is difference between optimal solution and solution found by these methods increase even more. From results is obvious that adapted IRS algorithm is not appropriate to solve the objective function $\sum D_j w_j$.

5. Efficiency of Scheduling Algorithms

Methods which can increase the efficiency of scheduling algorithms are described in this section. There is no a right general technique for writing an efficient code for scheduling algorithm. Implementation of algorithms is always depended on the given situation and on the requirement algorithm. But it is necessary to realize what the algorithm will solve, how will be able to extend and for what kind of problems the algorithm is determined. It is important to realize which part of the algorithm must be optimized (in term of code). The code profilers can help to this aim. Furthermore, a consecutive implementation of one scheduling algorithm in two different programming languages is contributing too. Mistakes can be found, code is thought over again and problem is seen from different point of view. Examples and general recommendations, which were registered implementation of scheduling algorithms, are described in this section.

5.1. Usage of Correct Programming Language

The scheduling algorithms are very time-consuming. There are many of algorithms which can find an optimal solution but there are not fast computers and their software environment which would find solution in tolerable time for hard problems. However, between programming languages and their usage are differences. Programming languages (and their advantages) which were used in this work are described in this section.

5.1.1. Matlab

"MATLAB - The Language of Technical Computing" is a high-level development environment for technical computing. This language is popular for fast and simple work with matrices, data handling and transparent evolution of algorithms. Add-on toolboxes (collections of special-purpose MATLAB functions, available separately) extend the MATLAB environment to solve particular classes of problems. We use Torsche Scheduling Toolbox as a support for development and verification of algorithms.

Torsche Scheduling Toolbox

TORSCHE Scheduling Toolbox for Matlab[1] is a freely (GNU GPL) available toolbox. This toolbox can be used for a complex scheduling algorithms design and verification. Graphs and schedules can be created and printed very easily with Torsche. Basic scheduling and graph algorithms are implemented in this toolbox too. Torsche is developed at the Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Control Engineering.

5.1.2. C++

C++ is a very popular programming language. The following advantages are important for us. This programming language is very fast. Licenses are no required for algorithms development in C++. The Programs implemented in this language can be compiled for different platforms. Many of libraries are accessible for this language. For these reasons, C++ was chosen as one of the programming language for final implementation of scheduling algorithms. Because work with matrices is somewhat complicated in C++, BOOST library and STL library were used. Many methods, which work with matrices and vectors, are implemented in these libraries. Small documentation was created for much using operations. This documentation describes transferring function from Matlab to C++.

STL (Standard Template Library)

STL[2] is a generic collection of class templates and algorithms that allow programmers to easily implement standard data structures like vectors, lists, queues and stacks.

BOOST

The Boost C++ Libraries[3] is a collection of many libraries that extend the functionality of C++. We used Graph and uBlas libraries for this work. They are generic classes, in the same sense as the Standard Template Library (STL). The Graph library is a generic interface that allows access to a graph's structure and basic graph algorithms. uBLAS provides matrix and vector classes as well as basic linear algebra routines. The uBLAS covers the usual basic linear algebra operations on vectors and matrices: addition and subtraction of vectors and matrices and multiplication with and the like.

[1] http://rtime.felk.cvut.cz/scheduling-toolbox
[2] http://www.sgi.com/tech/stl
[3] http://www.boost.org

5.1.3. C#

C# is a programming object oriented language designed around 1999 or 2000 by Anders Hejlsberg at Microsoft. C# is intended to be a simple, modern, general-purpose, object-oriented programming language. C#, in contrast to C++, include strong type checking, array bounds checking, detection of attempts to use uninitialized variables, source code portability, and automatic garbage collection. These aspects escalated robustness, durability and programmer productivity.

Extended libraries for work with matrices and graphs are available for C# too. Math.NET Iridium[1] and dnAnalytics[2] are similar libraries as BOOST (C++) as well QuickGraph[3] is similar library as Graph (C++) libraries. But basic equipment C# is sufficient to these requirements and for that reason there is no need to use these libraries. Besides, when we implemented algorithm with extended libraries we concentrated primarily on the exact transcript of algorithm from Matlab to C# (or C++). Furthermore, extended libraries must be installed to computer, their versions must be verified and their wrong application can lead to decrease of performance of scheduling algorithms.

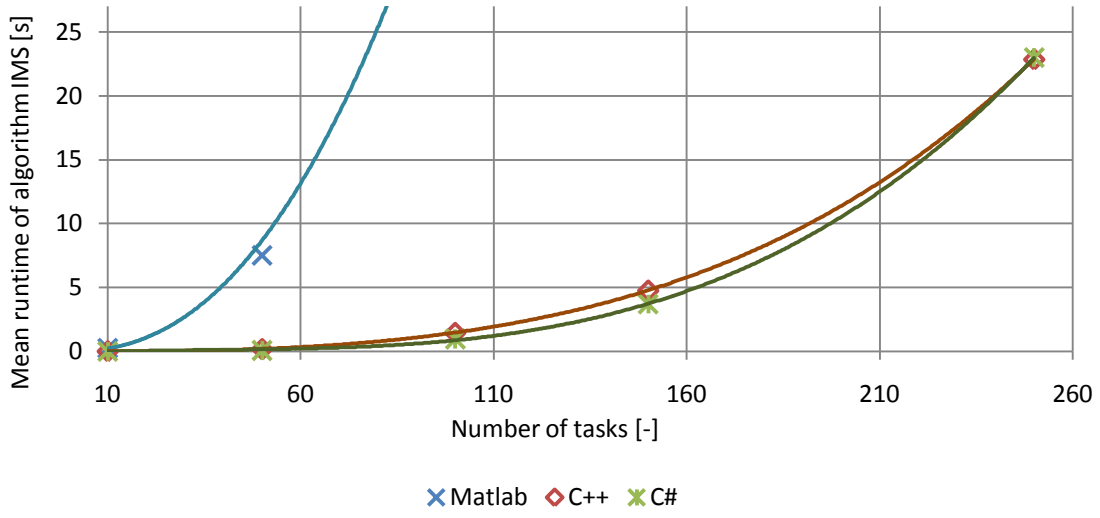


Fig. 14. Runtime comparison of IRS algorithm which is implemented in different programming languages.

There are many discussions between programmers that C# is a slower programming language then C++. For that reason both programming languages were compared. A runtime comparison of algorithm IRS which is implemented in different programming languages is showed In Fig. 14. It is seen that C++ and C# solve the same problems in the similar time. On the other hand, Matlab is much slower then C++ and C#.

For all these reasons it was decided that for final implementation C# programming language will be used.

[1] http://mathnet.opensourcedotnet.info
[2] http://www.codeplex.com/dnAnalytics
[3] http://www.codeplex.com/quickgraph

5.2. Integer Linear Programming

Integer Linear Programming (ILP) is not a programming language but a mathematical method for optimization of an objective function, subject to constraints formed by linear inequalities. Some scheduling problem can be described as linear inequality constraints. This description is sometimes simple as any scheduling algorithm and extra solution acquire by ILP is an optimal solution. Unfortunately, these methods are usable for small instances only. ILP is a favorite method and there are exist many solvers for this problem. The time comparison of ILP solvers is in Fig. 15. ILP problem formulation is used to find optional solutions for small instances in this work.

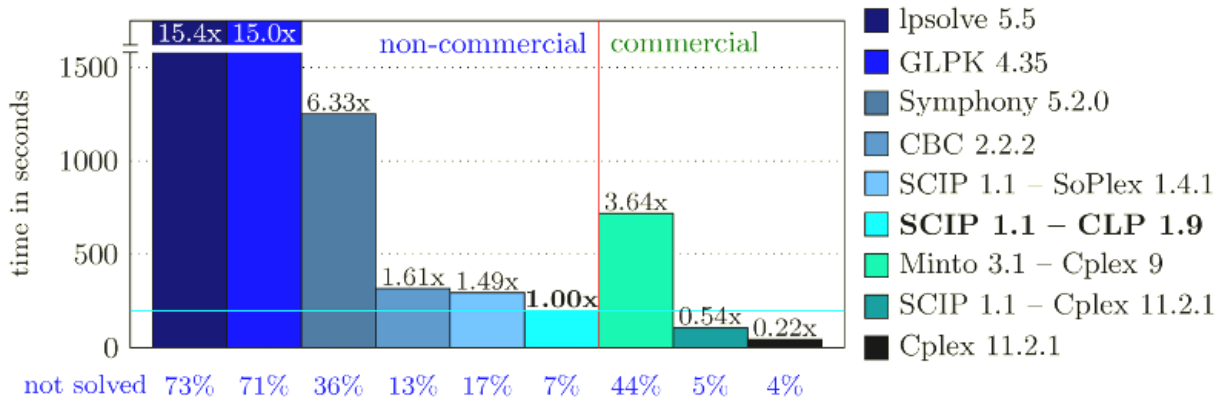


Fig. 15. Time comparison of ILP solvers from page: http://scip.zib.de/.

5.3. Usage and Suitable Format for Variables

They are many ways how variables can be stored and how with data can be manipulated. Some examples which can lead to improved work with data are shoved in this section.

When reading of data or pass variables between function is often repeated, runtime of algorithm can increase. If some variables are created as global variables, this effect may be defeated. Global variables can be profitably applied to often repeated calling of a function or to a recursive routine for example. Similarly, place for allocation of variables must be right chosen. Even we should think whether all of used variables are needed or not. Comparison of algorithm, where the rules described above were aplicated, is shoved in Fig. 16. Apropos, we find out that using extended libraries can lead to degradation of efficiency during aplicated this rules. For that reason it was decided that this libraries will not used in final implementation of algorithms.

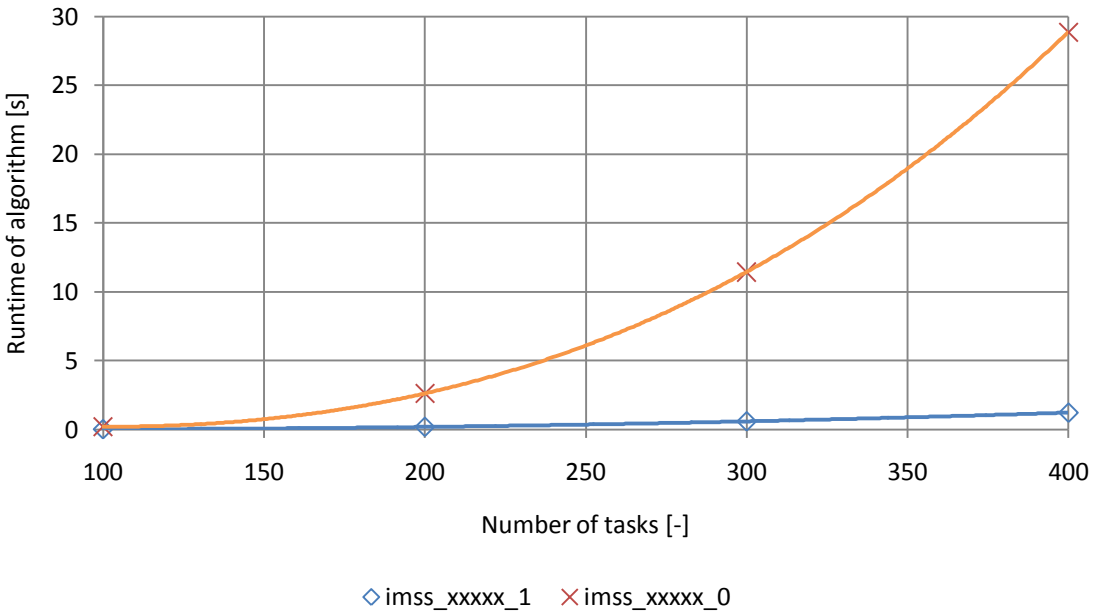


Fig. 16. Time comparison of algorithm with wrong allocating of variable (imss_xxxxx_0) and algorithm where were dressed using of variables (imss_xxxxx_1).

Data are often stored in matrixes or vectors in scheduling. Matrix which has very few elements is shoved in Fig. 17 (presuppose, zero elements are not important for us). Reading of all elements of this matrix takes $n \times m$ steps, where n is number of rows and m is number of columns in the matrix. If so called *sparse matrix* is created from this matrix, the number of steps will be decreased for reading of all elements. Sparse matrix in our example have size $3 \times l$, where l is number of records in the original matrix. The first column corresponds to indexes of rows in the original matrix, the second columns corresponds to indexes of columns in the original matrix and the third column corresponds to values from the original matrix. This sparse matrix is read in l steps. Similar matrix was used for representation the take-give resources in FBS algorithm.

$$\begin{bmatrix} 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 4 & 3 \\ 3 & 1 & 7 \\ 4 & 2 & 1 \end{bmatrix}$$

Fig. 17. A matrix and corresponding sparse matrix.

Next example shows that variable can be used not only as store of values. The variables can express state of algorithm too. A matrix is used by two of ways in our example. An original matrix is used in the first case and a transposed matrix is used in other case. A Program branching (*if* or *case* condition) can be an onus in the most time critical part of algorithm. This situation can be resolved as follows. Instead of two matrices with size $n \times m$ (original matrix and transposed matrix), we define one matrix with size $n \times m \times 2$. The values are addressed by three indexes in new matrix: i (from 1 to n) is index of row, j (from 1 to m) is index of column and k (1 or 2) represent matrix from which is read. One global variable is sufficient to address a right matrix. This method was used in FBS algorithm.

5.4. Look Ahead Counting Sub-results

Look ahead counting sub-results (or constants) are useful methods for acceleration of algorithm. If we want make any partial calculations in a loop (or in repeated calling of function) then it is faster this partial calculations count at first and then enter into the loop (if it is possible). It can be said generally, what was counted once needn't be counted again. If a function has not many of inputs/outputs parameters, it is possible to count all results (for all of inputs) before. A table can be used instead of the function. Similar example will be shoved in the following paragraph.

The objective of our function is to verify whether two tasks are overlapped or not. Two tasks are overlapped when both tasks are performed in the same time on the same processor.

If each task is performed on one processor only, the verification is trivial, i.e. we can compare numbers of processors only. When the numbers are the same, it can be verified whether tasks are performed in the same time. Contrariwise, when the numbers of the processors are different, it is not needed to compare whether the tasks are overlapped.

Indexes of task	Dedicated processors
1	1, 3, 4, 5
2	1, 4
3	2, 3, 5
4	3, 5

$$\begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Fig. 18. Table of dedicated processors and matrix which shows where tasks have the same processors.

But when multiprocessors task are considered, verification whether two tasks are on the same processor is complicated now. We must verify whether both of tasks have at least one same processor. If they have, it can be verified whether tasks are performed in the same time. The numbers each of processor of first task must be compared with the number each of processor of the second task. The more processors have a task the slower verification is.

It is advantageous to create a matrix, which keeps information about the tasks that are processed on the same processor. Such matrix is shoved in Fig. 18. Indexes of rows and columns are accorded to indexes of tasks. The values are accorded to information whether tasks are processed on the same processor or not, 1 or 0. The same matrix is used in algorithms IRS and FBS.

5.5. Usage of Previously Found Results

Usage of previously found results can be very useful method. Runtime of algorithm can be decreased, but also an objective function can be better. Three examples, using previously found results, are shoved in the following paragraphs.

Computation of arithmetic mean is the simplest example, but using of previously found results can be well shoved on this example. If we want to count the arithmetic mean repeatedly (e.g. values are stepwise increased) then it can be counted in two ways. A) All values must be available and arithmetic mean can be repeatedly counted them, according to the following formula

$$\overline{x_n} = \frac{\sum_{i=1}^{n} x_i}{n}$$

or B) The better formula can be used:

$$\overline{x_n} = \frac{\overline{x_{n-1}}(n-1) + x_n}{n}$$

We need count neither the sum of all values nor store these values in this case. We need to know only previous mean and number of values. This formula was used in IRS algorithm (TWT objective function).

This example is similar to the first example. The longest paths between all tasks were necessary counted repeatedly. The Bellman-Ford algorithm (see e.g. Brucker P., Knust S. 2005) was used to solve this problem initially. This algorithm requires $\mathcal{O}(n^2 m)$ time, where n and m are the number of tasks and the number of time lags respectively. Bellman-Ford algorithm is a relatively fast algorithm, however, the shortest paths are always counted from the algorithm beginning. This algorithm was replaced by an incremental algorithm Bartusch et al. (1988). Incremental algorithm updates distance matrix in $\mathcal{O}(n^2)$. Incremental algorithm is faster than Bellman-Ford algorithm since it uses previous results. This algorithm is described in Section 3.2.3 in detail.

How previously results can be used to improve an objective function is described in this paragraph. Heuristic algorithm IRS schedules tasks stepwise according to their priorities, i.e. from the task with the highest priority till the task with the smallest priority. The priority of task is determined according to the distance between this task and the latest (dummy) task in graph of precedence constraints. These priorities are counted at the beginning of the algorithm. The instance can be infeasible because sequence in which tasks were scheduled was wrong. The information, which tasks were removed from the schedule the most often, is stored during the algorithm running. If the original priorities of tasks are changed for the benefit of the most unscheduled tasks and the algorithm is started again, then a feasible schedule can be found. Thanks to this method it can be found more feasible schedules.

6. Experimental Results

Experimental results of IRS and FBS algorithms are discussed in this chapter. Instances which are used to benchmark of the algorithms are presented in the Section 6.1. The Sections 6.2. and 6.3. show the influence of set of parameters of algorithms on their results. Benchmark of both algorithms is shoved in section 6.4. and the experiments with time symmetry mapping are presented at the end of this chapter.

6.1. Instances and Implementation

Basic concept of all algorithms was implemented in Matlab. After that, the algorithms were transferred to C#, while the objective was the code efficiency (see Chapter 5). Both algorithms were compiled as COM Components under C#. All experiments were performed on Intel Pentium 1.66 GHz, 2 GB RAM.

Three types of instances were used to benchmark IRS and FBS algorithms. Generator of instances GEN_INS (Hanzálek and Šůcha 2009) was used for evolution and basic benchmark of algorithms. This generator allows transparently set many parameters of instance. The finally experiments were performed on standard instances generated by ProGenMax[1] and instances of a lacquers production (Behrmann et al. 2005).

6.1.1. GEN_INS

Generator GEN_INS was developed at the Czech Technical University in Prague, Faculty of Electrical Engineering, Department of Control Engineering. This generator creates the instances for $PS|temp, o_{ij}, tg|Cmax$ problem. The parameters and typically set of generator are shoved below.

Number of tasks	n
Maximal processing time	12
Number of positive time lags (edges with positive weight)	3*n/2
Number of negative time lags (edges with negative weight)	n/2

[1] http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax/rcpspmax.html

Maximal weight of positive time lags	15
Maximal weight of negative time lags	40
Maximal changeover time	8
Maximum of dedicated processors	2
Maximal number of multiprocessor tasks	2
Maximal capacity of processors	8
Number of take-give resources	3
Maximal number of groups of take-give resources	2
Maximal capacity of take-give resources	2

6.1.2. ProGenMax

ProGenMax generator is accessible on web site[1] of Universität Karlsruhe (TH) - Institute for Economic Theory and Operations Research. The instances generated by ProGenMax are used to benchmark different scheduling algorithms. These instances are stored in packages, each package includes 90 instances. For each instance are available value of the best result (C_{max}^{best}) and name of $algorithm$ which found this result: BB: Branch-and-bound algorithm, AM: Approximation method, FB: Filtered Beam Search, PR: Multi-Pass Priority-Rule Method, TS: Tabu Search, GA: Genetic Algorithm. Our experiments were preformed on packages UBOxxx (where xxx means number of tasks n in one instance) for $PS|temp|Cmax$ problem.

6.1.3. Lacquer Production

Lacquer production (Behrmann et al. 2005) is a real production scheduling problem. Lacquer production can be described as a project scheduling problem with general temporal constraints, resource constraints and take-give resource ($PS|temp, o_{ji}, tg|Cmax$).

Production line produces three different lacquers, universal lacquer (uni), metallic lacquer (*met*) and bronze lacquer (*bro*). Each lacquer has different manufacturing process and used different resources, see Fig. 19. Each instance for scheduling is determined by number of individual lacquer.

[1] http://www.wior.uni-karlsruhe.de/LS_Neumann/Forschung/ProGenMax

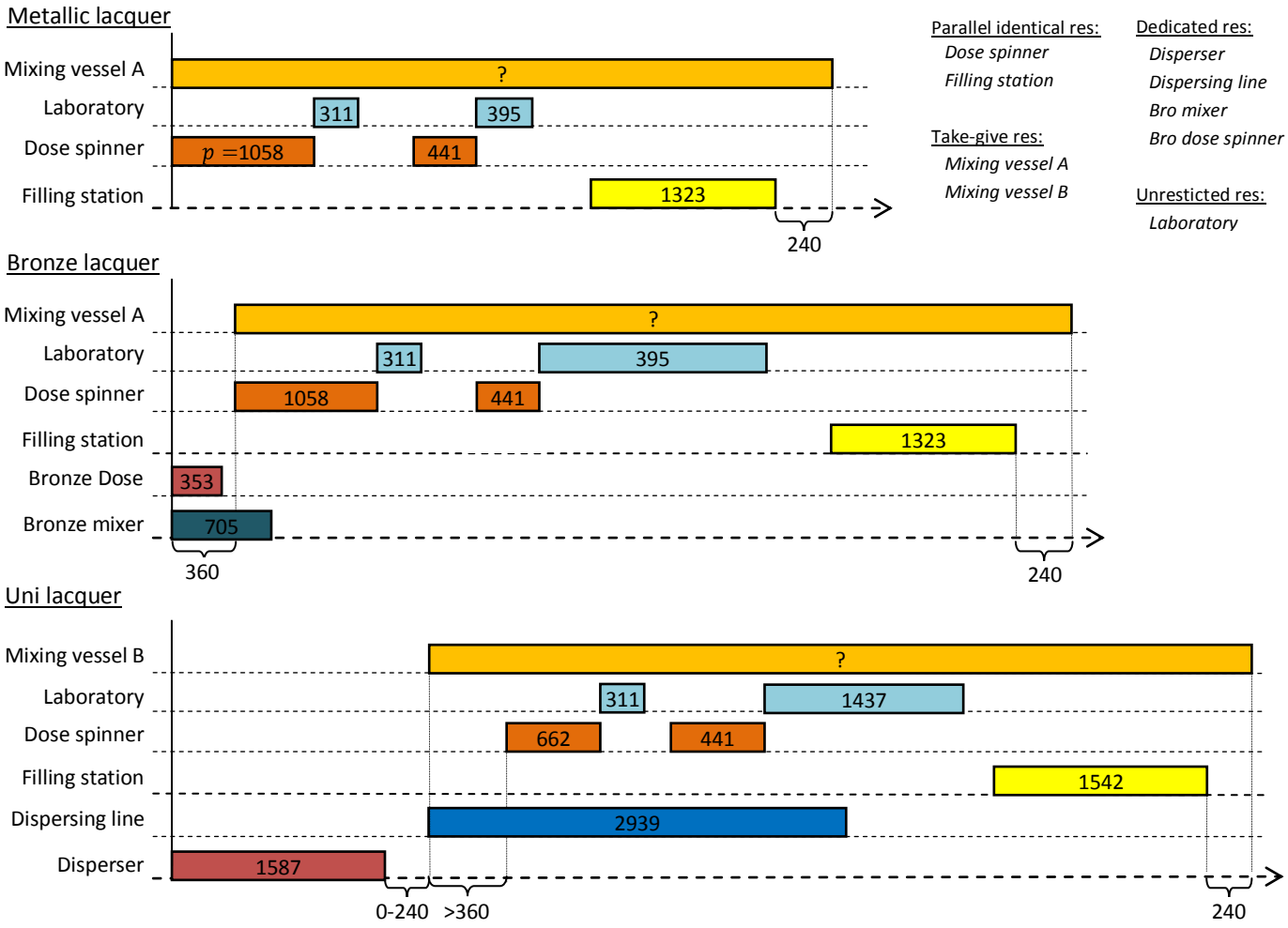


Fig. 19. Manufacturing process of lacquer production.

6.2. Parameters of IRS algorithm

Influence of algorithm parameter Budget Ratio and interval bisection method on the results found by iterative resource search algorithm is shoved and discussed in the next subsections.

6.2.1. Budget Ratio

The parameter $BudgetRatio$ is the ratio of the maximum number of activity scheduling steps to the number of tasks n. Influence of this parameter on the runtime of algorithm and number of feasible schedules is shown in Fig. 20. This experiment was performed on 90 benchmark instances UBO100 (with n=100). From the experiments follow that reasonable compromise between computation time and quality of the resulting schedule is usually achieved with $BudgetRatio = 2$.

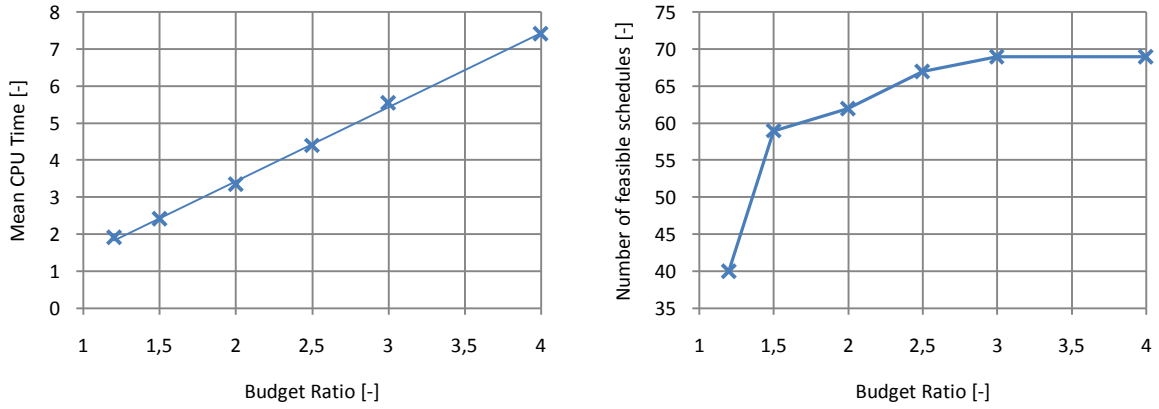


Fig. 20. Influence of Budget Ratio on the runtime of algorithm and number of feasible schedules.

6.2.2. Interval Bisection Method

Interval bisection method (binary search algorithm) is used, for example, for searching an approximate solution of equations or sampling of signals in electrotechnics. It is assumed that the values are sorted (in decreasing or increasing line) in the interval. The binary search method is base on verifying whether wanted value is greater than or less than middle of interval. Lower or upper bound of interval is changed to value of middle interval according previous result. The cycle is repeated until wanted value is not found. Time complexity of this method is $\mathcal{O}(\log n)$, where n is the number of values in the interval (interval on which searching is used). For comparison, time complexity of linear search is $\mathcal{O}(n)$.

Earlier version of IRS algorithm IRS evaluates lower and upper bound $Cmax$ $\langle LB, UB\rangle$ for given instance at first and then *findSchedule* function is called for all values from interval $\langle LB, UB\rangle$ until a feasible solution is found. Estimated $Cmax$ is an input parameter for *findSchedule* function. However, the interval $\langle LB, UB\rangle$ is different for each instance. Mean runtime of IRS is dependent on

time when feasible solution is found (see Table 3). If the instance is not feasible then the function *findSchedule* is called for all values from $\langle LB, UB \rangle$. For that reasons binary search method is used in IRS algorithm for found the best $Cmax$.

Table 3

Comparison IRS algorithm with binary search of estimate $Cmax$ and with linear search of $Cmax$.

	50 feasible instances		50 unfeasible instances		Common difference of $Cmax$
n tasks [-]	Runtime of IRS with bisection method [s]	Runtime of basic IRS algorithm [s]	Runtime of IRS with bisection method [s]	Runtime of basic IRS algorithm [s]	Mean (Cmax1-Cmax0) [%]
50	0,002	0,005	0,002	0,015	1,3
100	0,010	0,029	0,012	0,112	1,1
200	0,061	0,190	0,068	0,834	1,8
300	0,185	0,548	0,202	2,750	2,2
400	0,417	1,116	0,453	6,395	8,6
500	0,861	2,282	0,922	13,102	1,3

100 instances were generated for each number of tasks. Values are mean from all values.

But there is one problem here. IRS algorithm has not equalization of feasibility on interval $\langle LB, UB \rangle$. It is not true that the function *findSchedule* finds always a feasible schedule from some C. It means for as that binary search can lead to worst results. However, results are very good whereas acceleration of algorithm is huge (see Fig. 21 and Table 3).

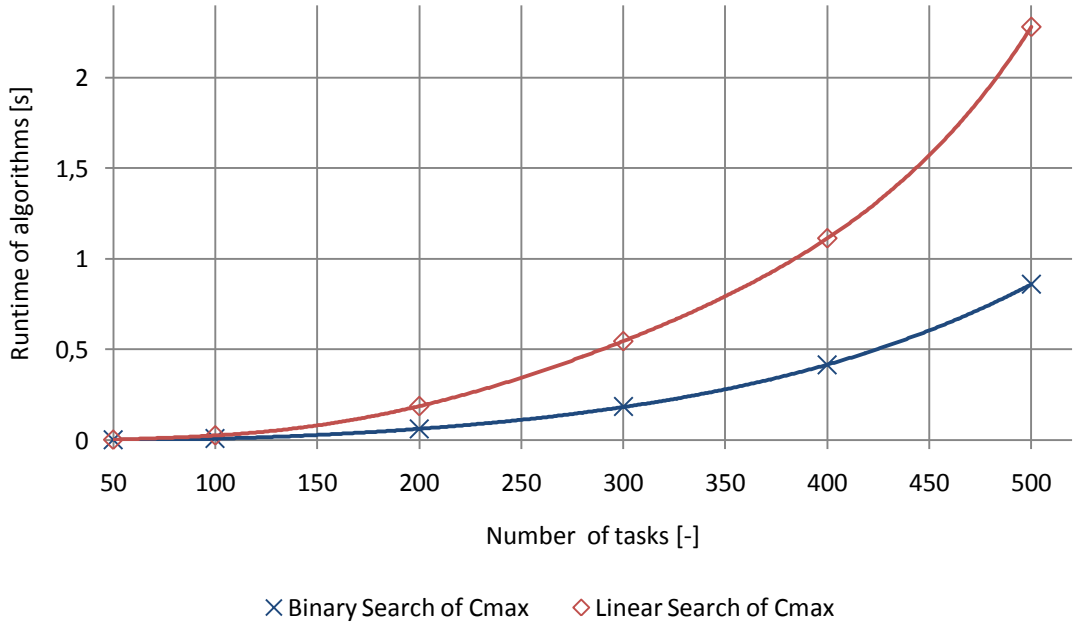


Fig. 21. The time comparison of algorithm IRS with binary search of $Cmax$ and algorithm IRS with linear search of $Cmax$.

6.3. Parameters of FBS algorithm

Influence of algorithm parameters filter width and beam width on the results found by filtered beam search algorithm is shoved and discussed in the following subsections.

6.3.1. Filter Width and Beam Width

Efficiency of filtered beam search method is discussed in this section. FBS algorithm has two main parameters, the filter width α and the beam width β. Both parameters have direct impact to the objective function and runtime of the algorithm too.

Filter width corresponds to the number of candidates which can be added to the tree of result. The more of candidates are evaluated by beam criterion the better candidate can be chosen to add to tree of result. Of course, evaluation more candidates cost much time. Comparison result for different filter width is shoved in Table 4. On the basis the results in Table 4, it was decided that filter width $\alpha = 3$ for next testing.

Table 4

Comparison of result for different filter width α for fbsm_xxxxx algorithm is shoved in table.

n tasks [-]	Mean Time [s]	Mean $Cmax$ [-]	Feasible schedules [-]	α	n tasks [-]	Mean Time [s]	Mean $Cmax$ [-]	Feasible schedules [-]	α
8	0,26	53,11	114	6	10	5,07	60,8	110	6
8	0,21	53,08	114	5	10	3,41	60,84	110	5
8	0,16	53,08	114	4	10	1,93	60,81	110	4
8	0,11	53,08	114	3	10	0,96	60,94	109	3
8	0,07	53,2	111	2	10	0,43	60,59	101	2

Values in table are mean from measurement of 200 instances.

Beam width β has similarly impact to results as filter width. Beam width determines number of new nodes in the result tree. Then Beam width determines the state space of algorithm. Of course, optimal solution of instance can be found easily if state space is bigger. Simultaneously, the runtime of algorithm grows up with the state space. Comparison between the constant beam width and the randomly evaluated beam width is shoved in Table 5. From the table follows that results of the algorithm are dependent on its set. The algorithm runtime, with β chosen from interval $\langle 1,2 \rangle$ randomly, is much smaller then algorithm with $\beta = 2$. For this reason $\beta = 2$ is not used for bigger instance.

Table 5

Comparison the constant beam width and the randomly evaulated beam width.

	$\beta = 2$			$\beta = [1-2]$ (randomly)		
n tasks [-]	Mean Time [s]	Mean $Cmax$ [-]	Feasible schedules [-]	Mean Time [s]	Mean $Cmax$ [-]	Feasible schedules [-]
12	0,10	69,33	105	0,01	69,69	104
15	4,61	78,27	100	0,02	77,91	94
20	46,68	86,17	90	3,15	85.40	89

Values in table are mean from measurement of 200 instances.

6.3.2. Time Limit for Runtime of Algorithm

State space of tree of results can be reduced by algorithm parameter beam width β. But runtime of algorithm is enormously increased for bigger instances. For that reason, maximal runtime of the algorithm is the next input parameter of FBS algorithm. This limit is usually 60 second. Influence of this parameter on the results is shoved in Table 6.

Table 6

Infuence of maximal runtime of FBS algorithm to results.

n tasks [-]	Mean Time [s]	Mean C_{max} [-]	Feasible schedules [-]	Max Runtime [s]
12	0,048	69,5	105	1
12	0,093	69,5	105	10
12	0,097	69,3	105	60
12	0,097	69,3	105	∞

n tasks [-]	Mean Time [s]	Mean C_{max} [-]	Feasible schedules [-]	Max Runtime [s]
15	0,180	78,6	100	1
15	0,739	78,3	100	10
15	2,089	78,0	100	60
15	4,614	78,3	100	∞

Values in table are mean from measurement of 200 instances.

6.4. Benchmark of Algorithms

Comparison of FBS algorithm and IRS algorithm on ProGenMax instances is shoved in Table 7. From results follow that FBS algorithm reaches worse results than IRS algorithm. The number of feasible schedules found by FBS algorithm is very decreasing for instances with more taks. Algorithm FBS needed often the full time limit of runtime algorithm to find feasible schedules (60 second), but time limit was for many instances too small.

Generally, IRS algorithm reaches better results than FBS algorithm. Runtime of algorithm is small, number of feasible schedule is not critically decreasing (with gowning number of tasks) and even mean $Cmax$ is smaller than mean C_{max}^{best}.

Table 7

Comparison of FBS algorithm and IRS algorithm on ProGenMax instances.

		FBS			IRS			ProGenMax	
Package	**n tasks [-]**	**Mean Time [s]**	**Mean C_{max} [-]**	**Feasible schedules [-]**	**Mean Time [s]**	**Mean C_{max} [-]**	**Feasible schedules [-]**	**Mean C_{max}^{best} [-]**	**Feasible schedules [-]**
UBO 10	10	0,127	50,3	69	0,043	49,4	73	51,9	73
UBO 20	20	15,323	94,0	62	0,136	91,3	62	104,1	70
UBO 50	50	50,654	201,5	53	0,769	182,8	61	188,7	73
UBO 100	100	56,148	339,1	23	3,141	307,1	62	362,9	78

Each package includes 90 instances.

Comparison of FBS algorithm and IRS algorithm on Lacquer production instances is presented in Table 8. Contrary to the previous results FBS algorithm is usable for scheduling of large instances too. It flows from base of FBS algorithm. FBS algorithm resolve conflicts for tasks with take-give resources at first and then it resolve of conflicts between tasks. In other words, FBS algorithm approximately schedules individual lacquers at first and then precisely schedules individual tasks.

By comparison both algorithms, it can be seen that they have similar results. Runtime of algorithm IRS grows for a large order of lacquer, but the runtime is still in the bounds.

Table 8

Comparison of FBS algorithm and IRS algorithm on Lacquer production instances.

Orders			**n tasks [-]**	**FBS**		**IRS**	
uni	**met**	**bro**		**CPU Time [s]**	**C_{max} [-]**	**CPU Time [s]**	**C_{max} [-]**
2	2	2	46	0,08	17 610	0,08	21 398
5	5	5	115	57,66	48 665	0,58	49 255
10	10	10	230	60,03	98 871	5,14	94 562
15	15	15	345	60,04	147 729	17,42	148 391
20	20	20	460	60,07	196 246	51,7	187 841

One instance was generated and tested for each order.

6.5. Experiments with Time Symmetric Mapping of Instances

In order to improve the solution while using the same heuristic algorithms we use the time symmetry mapping to find a new feasible solution and to improve the objective function (C_{max}). If we want to create a schedule in backward time orientation, we use TSM to instance before start of an algorithm and then this (reversed) instance is input parameter of the algorithm. Thus the same algorithm is used to schedule for forward and backward time orientation of the problem (Hanzálek and Šůcha 2009).

Time symmetric mapping is used differently in this work too. TSM is implemented inside the body of the algorithms. Algorithms schedule several steps with the forward orientation of the problem and several steps with the backward orientation of the problem. Description of the implementation of TSM IRS and FBS algorithm is presented in Section 3.1.2 and 3.2.4.

Influence of time symmetric mapping to results of FBS and IRS algorithm is showed in Table 9 and Table 10. Instances generated by ProGenMax generator was used as a benchmark. Number of feasible schedules found (feasibility), difference C_{max} and runtime of the algorithm are compared in the tables.

The best results of different algorithms, which are enclosed to instances ProGenMax, are showed in the column with label $Best$. The difference C_{max} is calculated as difference between found solutions and the corresponding best solutions. Thus, found solutions are compared with the best solutions on these instances. Next columns of the table show results of FBS and IRS algorithm for the original problem (F forward), for the problem obtained by TSM (BW backward) before start of the algorithms and for TSM implemented inside the body of the algorithms (AXX). A value XX correspond to number of scheduling steps after which instance is reversed. Columns F+BW and ALL include the best results for the original problem + for problem obtained by TSM before the start of the algorithm respectively the best results from all five methods of TSM usage. Results of TSM for FBS algorithm and IRS algorithm application are described in next two paragraphs.

Table 9
Influence of time symmetric mapping to results of FBS algorithm.

Feasibility [-]								
Package	**Best**	**F**	**BW**	**A10**	**A20**	**A50**	**F+BW**	**ALL**
UBO100	78	23	21	16	18	23	28	37
Difference C_{max} [%]								
Package	C_{max}^{best}	**F**	**BW**	**A10**	**A20**	**A50**	**F+BW**	**ALL**
UBO100	0	9,4	7,7	13,0	13,1	11,6	8,6	11,1
Mean CPU Time [s]								
Package	**Best**	**F**	**BW**	**A10**	**A20**	**A50**		
UBO100	max 100	50,96	54,53	58,77	56,43	53,75		

Each package includes 90 instances. For detail see Table 14.

We can see that influence of TSM to feasibility of FBS algorithm is not visible immediately (see Table 9). Only one variant of usage of TSM (A50) found the same number of feasible schedules as the scheduling without TSM. But when we observe the results in detail in Table 13, we can see that each variant finds out feasible solutions for different instance. Thus when we count these different results (see summarize columns F+BW and ALL), we get 5 respectively 14 new feasible solutions thanks to the TSM. Difference C_{max}^{best} and other C_{max} is bigger for TSM implemented inside FBS algorithm than for basic variants of experiments. The FBS algorithm was tested on instances with 100 task only. The reason were the bad results of FBS algorithm (too small number of feasible solutions and relatively big C_{max}).

The influence of TSM to results of IRS algorithm is presented in Table 10. We can see similar effect of TSM on results as in FBS algorithm. Number of feasible solutions is bigger and C_{max} increases for TSM used inside the algorithm. But IRS algorithm finds much more feasible results, results with smaller C_{max} and with smaller runtime of algorithm than FBS algorithm. Moreover, IRS algorithm found 10 better schedules than best results from package UBO500, see Table 15. From tests of both algorithms

follows that TSM has no influence on runtime of the algorithms. The runtime is similar even if the TSM is implemented inside the algorithms.

These tests show that TSM is simple method how we can get more feasible solutions with smaller objective function with the same (or a little modified) algorithm. If we want use this method in practice, we must repeatedly run the algorithm for different set of TSM (in the same way as in the tests above). The total runtime of the algorithm would be bigger of course. But if we would use, for example, a computer with more processors, we would get results in the same time.

Table 10
Influence of time symmetric mapping to results of IRS algorithm.

Feasibility [-]								
Package	**Best**	**F**	**BW**	**A16**	**A32**	**A64**	**F+BW**	**ALL**
UBO100	78	62	71	72	70	69	73	75
UBO500	79	59	61	57	60	60	63	64
Difference C_{max} [%]								
Package	C_{max}^{best}	**F**	**BW**	**A16**	**A32**	**A64**	**F+BW**	**ALL**
UBO100	0	3,3	5,3	7,4	6,9	6,2	3,7	4,1
UBO500	0	1,2	1,0	2,3	2,8	2,3	1,3	1,6
Mean CPU Time [s]								
Package	C_{max}^{best}	**F**	**BW**	**A16**	**A32**	**A64**		
UBO100	max 100	3,44	3,76	2,39	2,57	2,81		
UBO500	max 500	60,1039	68,3965	72,76	67,78	65,23		

Each package includes 90 instances. For detail see Table 13 and Table 15.

7. Conclusions

Two scheduling algorithms were implemented in this work. These algorithms are capable to solve general and real scheduling instances for problem $PS|temp, o_{ij}, tg|C_{max}$. The scheduling algorithms are very time consuming and therefore implementation of these algorithms was concentrated on the code efficiency.

Iterative resources scheduling algorithm (IRS) reaches very good results on all tested types of instances. This algorithm is able to find very good results in short time on general and real scheduling problems. IRS algorithm is able to schedule large instances. Instances with 500 tasks are tested in this work, instances with 1000 was tested in Hanzálek and Šůcha 2009. IRS algorithm reaches comparable results (in some cases even better) as the best results of algorithms, which were tested on the standard benchmarks for $PS|temp|C_{max}$ problem (ProGenMax instances). This algorithm will be presented on multidisciplinary international scheduling conference (MISTA 2009).

Two scheduling algorithms based on different principles were chosen to benchmark IRS algorithm. After studying these algorithms, it was decided that filtered beam search algorithm (Schwindt and Trautmann 2003) will be implemented. This algorithm was created for real scheduling problem, rolling ingots production. FBS algorithm did not reach very good results on the general instances. But this algorithm was partially comparable for scheduling of real scheduling problem, e.g. lacquer production. Filtered beam search method is based on the depth-first search. The tree of solution is growing rapidly with number of resources conflicts and resources conflicts is growing rapidly with number of tasks in instance. For that reason runtime of FBS algorithm is too large and results are worse.

On the basis of skills with implementation of these algorithms was created a set of general recommendations how to implement efficient scheduling algorithms. These recommendations were presented on real examples.

Moreover, IRS algorithm was modified to minimize another objective function, i.e. total weighted tardiness. But this adaptation was stopped in the process of evolution, because IRS algorithm is not very suitable for minimization of total weighted tardiness.

Further, influence of time symmetric mapping (TSM) to results was investigated in this work. TSM was tested outside and inside the body of both algorithms. This method was showed to be very useful. We can obtain new feasible solutions or solutions with smaller objective function thanks to this method. Moreover, TSM is relatively simple method and for its usage it is not needed editing algorithms or only very little.

8. References

Bartusch, M.; Mohring, R. H.; and Radermacher, F. J. (1988), Scheduling project networks with resource constraints and time windows. Annals of OR 16:201–240.

Behrmann G., Brinksma Ed, Hendriks M., Mader A. (2005). Production scheduling by reachability analysis – a case study. *Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, page 140.1. IEEE Computer Society Press, 2005.

Bierwirth C. (1995), A generalized permutation approach to jobshop scheduling with genetic algorithms. *OR Spektrum* 1995; 17:87–92.

Blazewicz J., Ecker K. H., Pesh E., Schmidt G., Weglarz J. (2001), Scheduling Computer and Manufacturing Processes. *Second edition, Springer.*

De Bontridder, K. M. J. (2005). Minimizing total weighted tardiness in a generalized job shop. *Journal of Scheduling 8*: 479–496, 2005

Brucker P., Knust S. (2005). Complex scheduling*, Springer.* ISBN: 13978-3-540-29545-7

Brucker P., Hilbig T., Hurink J. (1999). A branch and bound algorithm for a single-machine scheduling problem with positive and negative time-lags. *Discrete Applied Mathematics*, 94(1-3):77-99, May 1999.

Brucker P., Kampmeyer T. (2008). Cyclic job shop scheduling problems with blocking. *Annals of Operations Research*, 159(1):161-181, 2008.

Carroll, D. C., (2005), Heuristic Sequencing of Jobs with Single and Multiple Components, Ph.D. Thesis, *Sloan School of Management*, MIT

Caumond A, Lacomme P, Tchernev N. (2005), Feasible schedules generation with an extension of the Giffler and Thomson algorithm for the job-shop with timelags. In: *International conference on industrial engineering and system management*, Marrakech-Morocco, 2005.

Caumond A., Lacomme P., Tchernev N. (2007), A memetic algorithm for the job-shop with time-lags, *Computers & Operations Research 35 (*2008), 2331-2356.

Cesta A., Oddi A., Smith S. F. (2002). A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8(1):109-136, 2002.

Cicirello V. A., S. F. Smith (2004), Heuristic Selection for Stochastic Search Optimization: Modeling Solution Quality by Extreme Value Theory. *Springer-Verlag Berlin Heidelberg*, M. Wallace (Ed.): CP 2004, LNCS 3258, pp. 197–211.

Colak, A. B., Keha, A. B. (2007). Interval-indexed formulation based heuristic for single machine total weighted tardiness problem. *Computers & Operations Research,* August2008, Author's Accepted Manuscript.

Frank B., Neumann K., Schwindt Ch. 2001. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spektrum*, 23(3):297-324, August 2001.

Gadkari, A., Pfund, M. E., Fowler, J. W., & Chen, Y. (2007). Scheduling jobs on parallel machines with setup times and ready times, *Computers & Industrial Engineering 54* (2008) 764–782

Hanzálek Z., Šůcha P. (2009). Time symmetry of Project Scheduling with with Time Windows and Take-given Resources. *Multidisciplinary International Scheduling Conference*, Dublin, 2009.

Laborie P. (2003). Algorithms for propagating resource constraints in ai planning and scheduling: existing approaches and new results. *Artif. Intell.,* 143(2):151-188, 2003.

Lee, Y.H., Bhaskaran, K. and Pinedo, M. (1997). A Heuristic to Minimize the Total Weighted Tardiness with Sequence Dependent Setups. *IIE Transactions*, Volume 29, Issue 1 January 1997 , pages 45 - 52.

Logendran, R., McDonell, B., Smucker, B. (2006). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research* 34 (2007) 3420-3438.

Mascis A., Pacciarelli D. (2002). Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research,* 143(3):498-517, 2002.

Monch, L., Balasubramanian, H., Fowler, J. W., Pfund, M. E. (2004). Heuristic scheduling of jobs on parallel baatch machina with incompatible job families and unequal ready times. *Computers & Operations Research* 34 (2005) 2731-2750.

Neumann, K., Schwindt, C., Zimmermann, J. (2003). Project Scheduling with Time Windows and Scarce Resources. *Springer, Berlin,* ISBN 3540401253.

Park, Y., Kim, S., Lee, Y. H. (2000). Scheduling jobs on parallel machines applying neuralnetwork and heuristic rules. *Computers & Industrial Engineering 38* (2000) 189-202.

Pfund, M. E., Balasubramanian, H. J., Fowler, W., Mason, S. J., Rose, O. (2007). A multi-criteria approach for scheduling semiconductor wafer fabrication facilities. Published online: 27 November 2007, *Springer Science+Business Media*, LLC 2007.

Rachamadugu, R. V., Morton, T. E. (1981). Myopic Heuristics for the Single Machine Weighted Tardiness Problem.*Graduate School of Industrial Administration, Carnegie-Mellon University,* Working Paper #28-81-82.

Rau, B. R. (2000). Iterative modulo scheduling. *PROGRES 2000 Workshop on Embedded Systems*, Utrecht, The Netherlands, 2000.

Schwindt, C. and Trautmann, N. (2003), Scheduling the production of rolling ingots: industrial context, model, and solution method. *International Transactions in Operational Research,* res. 10 (2003) 547-563.

Smith, T. B., Pyle, J. M. (2004), An Effective Algorithm For Project Scheduling With Arbitrary Temporal Constraints. In: Proceedings of the 19th *National Conference on Artificial Intelligence*, 2004.

Terada J., Vo H., Joslin D., Combining genetic algorithms with squeaky wheel optimization. In GECCO '06: *Proceedings of the 8th annual conference on Genetic and evolutionary computation,* pages 1329-1336, New York, NY, USA, 2006.

Valente, J., Alves, R. (2008), Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. *Computer & Operations Research* 35 (2008) 2388-2405.

Vepsalainen, A., Morton, T. (1987). Priority rules for job shops with weighted tardiness costs. *Management science*, Vol. 33. No. 8. August 1987

9. List of the Figures

10. Appendix

Table 11
Data to Fig. 14.

	Mean runtime of algorithm IRS [s]		
n tasks [-]	Matlab	C++	C#
10	0,23	0,01	0,01
50	7,49	0,18	0,08
100	46,62	1,45	0,98
150	---	4,76	3,69
250	---	22,85	22,97

100 instances were generated for each number of task and values are mean from all values.

Table 12
Data to Fig. 16.

n tasks [-]	Runtime of algorithm imss_xxxxx_1 [s]	Runtime of algorithm imss_xxxxx_0 [s]
100	0,03	0,23
200	0,20	2,64
300	0,60	11,46
400	1,23	28,86

100 instances were generated for each number of tasks. Values are mean from all values.

Table 13

Time Symmetric Mapping inside of FBS algorithm

Instance	Algorithm	C_{max}^{best}	C_{max}^{FBS} [-]					Instance	Algorithm	C_{max}^{best}	C_{max}^{FBS} [-]				
			F	BW	A10	A20	A50				F	BW	A10	A20	A50
1	BB	inf	inf	inf	inf	inf	inf	46	FB	283	inf	inf	inf	inf	inf
2	BB	inf	inf	inf	inf	inf	inf	47	FB	302	inf	inf	**401**	inf	inf
3	BB	inf	inf	inf	inf	inf	inf	48	AM	433	inf	inf	inf	inf	inf
4	FB	429	inf	inf	inf	inf	inf	49	FB	203	inf	inf	**274**	inf	inf
5	BB	inf	inf	inf	inf	inf	inf	50	FB	269	inf	inf	inf	inf	inf
6	BB	inf	inf	inf	inf	inf	inf	51	BB	272	inf	**321**	inf	inf	inf
7	GA	447	inf	inf	inf	inf	inf	52	BB	304	326	**323**	330	339	353
8	FB	435	inf	inf	inf	inf	inf	53	BB	177	210	**182**	inf	inf	inf
9	BB	inf	inf	inf	inf	inf	inf	54	BB	352	**366**	366	inf	inf	374
10	GA	522	inf	inf	inf	inf	inf	55	AM	247	**261**	269	inf	266	308
11	GA	263	inf	inf	inf	inf	inf	56	AM	288	inf	**302**	inf	305	329
12	FB	224	inf	inf	inf	inf	inf	57	BB	356	inf	**411**	inf	415	422
13	GA	180	inf	inf	inf	inf	inf	58	BB	317	330	**317**	319	330	319
14	FB	206	inf	inf	inf	inf	inf	59	AM	256	inf	inf	inf	inf	inf
15	BB	275	inf	inf	inf	inf	inf	60	FB	188	225	237	inf	**224**	253
16	FB	144	inf	inf	inf	inf	inf	61	BB	680	inf	inf	inf	inf	inf
17	BB	287	inf	inf	inf	inf	inf	62	BB	540	inf	inf	inf	inf	inf
18	BB	306	inf	inf	inf	inf	**346**	63	BB	inf	inf	inf	inf	inf	inf
19	FB	200	inf	inf	inf	inf	inf	64	TS	538	inf	inf	inf	inf	inf
20	FB	209	inf	inf	inf	inf	inf	65	GA	451	inf	inf	**585**	inf	inf
21	AM	262	**317**	322	339	332	337	66	BB	inf	inf	inf	inf	inf	inf
22	BB	492	**522**	523	538	538	536	67	TS	459	inf	inf	inf	inf	inf
23	BB	269	inf	inf	inf	**311**	inf	68	BB	540	inf	inf	631	**662**	inf
24	AM	192	inf	**221**	inf	inf	inf	69	BB	inf	inf	inf	inf	inf	inf
25	BB	194	**226**	inf	240	inf	229	70	GA	422	inf	inf	inf	inf	inf
26	BB	178	204	204	inf	inf	**197**	71	BB	514	inf	inf	inf	inf	inf
27	BB	225	inf	inf	inf	inf	inf	72	BB	inf	inf	inf	inf	inf	inf
28	BB	240	269	**261**	inf	inf	261	73	BB	414	inf	**466**	inf	inf	inf
29	BB	284	306	**284**	inf	inf	293	74	BB	255	inf	inf	inf	inf	inf
30	BB	196	244	inf	inf	**223**	inf	75	BB	534	inf	inf	inf	inf	inf
31	BB	inf	inf	inf	inf	inf	inf	76	AM	411	inf	455	**451**	inf	inf
32	GA	485	inf	inf	inf	inf	inf	77	TS	351	inf	inf	inf	inf	inf
33	GA	435	inf	inf	inf	inf	inf	78	BB	412	inf	inf	inf	inf	inf
34	GA	488	inf	inf	inf	inf	inf	79	TS	483	inf	inf	inf	inf	inf
35	BB	inf	inf	inf	inf	inf	inf	80	BB	503	inf	inf	inf	inf	**552**
36	BB	457	inf	inf	inf	inf	inf	81	BB	453	480	inf	491	542	**473**
37	TS	453	inf	inf	inf	inf	inf	82	BB	571	inf	596	591	593	590
38	GA	483	inf	inf	inf	inf	inf	83	FB	243	**262**	283	inf	303	272
39	GA	462	**605**	inf	631	664	625	84	BB	237	inf	inf	inf	291	**282**
40	AM	504	inf	inf	inf	inf	inf	85	BB	497	**502**	503	520	526	512
41	PR	363	**421**	inf	inf	inf	inf	86	BB	531	inf	inf	**577**	inf	inf
42	FB	359	inf	inf	inf	inf	inf	87	AM	368	inf	inf	inf	inf	inf
43	AM	359	inf	inf	inf	inf	inf	88	BB	402	**428**	457	inf	inf	468
44	BB	491	inf	inf	inf	inf	inf	89	BB	374	409	**385**	inf	inf	437
45	BB	407	inf	inf	inf	inf	inf	90	BB	476	480	**479**	517	500	inf

Instances are from package UBO100. Time limit of runtime of FBS algorithm was set on 60 second.

Table 14

Time Symmetric Mapping inside of IRS algorithm

Instance	Algorithm	C_{max}^{best}	C_{max}^{IRS} [-]					Instance	Algorithm	C_{max}^{best}	C_{max}^{IRS} [-]				
			F	BW	A16	A32	A64				F	BW	A16	A32	A64
01	BB	inf	inf	inf	inf	inf	inf	46	FB	283	**321**	333	333	340	346
02	BB	inf	inf	inf	inf	inf	inf	47	FB	302	313	**312**	347	361	333
03	BB	inf	inf	inf	inf	inf	inf	48	AM	433	447	450	451	**442**	450
04	FB	429	**452**	486	497	481	535	49	FB	203	207	**203**	207	203	215
05	BB	inf	inf	inf	inf	inf	inf	50	FB	269	304	**295**	inf	308	303
06	BB	inf	inf	inf	inf	inf	inf	51	BB	272	inf	**286**	305	306	312
07	GA	447	inf	**471**	531	482	482	52	BB	304	**304**	304	304	304	304
08	FB	435	**452**	453	471	456	482	53	BB	177	**177**	177	177	177	177
09	BB	inf	inf	inf	inf	inf	inf	54	BB	352	**352**	352	352	352	352
10	GA	522	inf	inf	inf	inf	inf	55	AM	247	**247**	247	247	247	247
11	GA	263	281	352	**272**	284	293	56	AM	288	294	**289**	294	291	291
12	FB	224	249	**235**	284	inf	258	57	BB	356	374	**364**	374	374	374
13	GA	180	189	inf	187	197	201	58	BB	317	**317**	317	317	317	317
14	FB	206	**209**	229	221	229	230	59	AM	256	**256**	256	257	256	256
15	BB	275	**275**	275	275	275	278	60	FB	188	190	**188**	189	188	188
16	FB	144	**159**	165	190	172	166	61	BB	680	inf	**705**	728	726	727
17	BB	287	**287**	287	287	287	287	62	BB	540	inf	**588**	606	inf	inf
18	BB	306	307	**306**	306	306	306	63	BB	inf	inf	inf	inf	inf	inf
19	FB	200	inf	**234**	302	284	292	64	TS	538	**578**	584	590	586	594
20	FB	209	inf	245	243	257	**226**	65	GA	451	496	501	**495**	503	504
21	AM	262	**262**	262	262	262	262	66	BB	inf	inf	inf	inf	inf	inf
22	BB	492	**500**	508	508	510	504	67	TS	459	inf	inf	inf	inf	inf
23	BB	269	**269**	269	269	269	272	68	BB	540	569	579	**558**	590	562
24	AM	192	**192**	197	192	192	204	69	BB	inf	inf	inf	inf	inf	inf
25	BB	194	**194**	194	194	194	195	70	GA	422	**438**	478	457	462	479
26	BB	178	**178**	178	178	178	178	71	BB	514	**529**	547	537	542	542
27	BB	225	**231**	244	254	288	254	72	BB	inf	inf	inf	inf	inf	inf
28	BB	240	**240**	240	240	240	240	73	BB	414	**427**	477	448	inf	inf
29	BB	284	**284**	284	284	284	284	74	BB	255	inf	306	296	300	**295**
30	BB	196	**196**	196	196	196	196	75	BB	534	546	**536**	538	536	538
31	BB	inf	inf	inf	inf	inf	inf	76	AM	411	**419**	426	425	430	432
32	GA	485	inf	inf	**572**	597	inf	77	TS	351	384	**371**	394	391	393
33	GA	435	inf	**494**	561	579	inf	78	BB	412	**424**	453	435	440	437
34	GA	488	inf	inf	**563**	inf	inf	79	TS	483	485	515	**507**	510	508
35	BB	inf	inf	inf	inf	inf	inf	80	BB	503	529	523	539	538	**520**
36	BB	457	inf	inf	546	557	584	81	BB	453	**455**	458	466	470	474
37	TS	453	**500**	497	510	509	501	82	BB	571	582	**569**	590	579	579
38	GA	483	553	540	698	**535**	inf	83	FB	243	244	**243**	247	243	243
39	GA	462	**487**	512	531	555	508	84	BB	237	**237**	237	237	237	237
40	AM	504	inf	inf	inf	inf	inf	85	BB	497	501	501	502	**500**	501
41	PR	363	376	384	382	388	**370**	86	BB	531	539	533	532	532	**531**
42	FB	359	inf	376	399	404	**368**	87	AM	368	**372**	390	392	393	395
43	AM	359	**372**	380	inf	396	409	88	BB	402	408	**402**	409	407	413
44	BB	491	inf	**548**	inf	inf	551	89	BB	374	**374**	374	374	374	374
45	BB	407	439	435	415	429	**431**	90	BB	476	**477**	477	477	477	477

Instances are from package UBO100.

Table 15

Time Symmetric Mapping inside of IRS algorithm

Instance	Algorithm	C_{max}^{best}	C_{max}^{IRS} [-]					Instance	Algorithm	C_{max}^{best}	C_{max}^{IRS} [-]				
			F	BW	A16	A32	A64				F	BW	A16	A32	A64
1	BB	inf	inf	inf	inf	inf	inf	46	AM	821	**821**	821	821	821	821
2	GA	2353	inf	inf	inf	inf	inf	47	GA	1512	1423	**1412**	1413	1412	1425
3	GA	2045	inf	inf	inf	inf	inf	48	GA	1181	inf	inf	inf	**1508**	inf
4	TS	2774	inf	inf	inf	inf	inf	49	GA	1209	inf	**1383**	inf	inf	inf
5	BB	inf	inf	inf	inf	inf	inf	50	BB	1326	**1326**	1326	1326	1326	1326
6	GA	2558	inf	inf	inf	inf	inf	51	AM	1223	**1223**	1223	1223	1223	1223
7	BB	inf	inf	inf	inf	inf	inf	52	BB	1109	**1109**	1109	1109	1109	1109
8	GA	2212	inf	inf	inf	inf	inf	53	BB	1029	**1029**	1029	1029	1029	1029
9	BB	inf	inf	inf	inf	inf	inf	54	BB	825	**825**	825	825	825	825
10	GA	2366	inf	inf	inf	inf	inf	55	AM	1153	**1153**	1153	1153	1153	1153
11	AM	589	**589**	589	589	589	589	56	BB	976	**976**	976	976	976	976
12	BB	1101	**1101**	1101	1400	1101	1101	57	BB	1238	**1238**	1238	1238	1238	1238
13	AM	1424	1430	**1424**	1430	1430	1427	58	BB	1314	**1314**	1314	1314	1314	1314
14	GA	1130	inf	inf	inf	inf	inf	59	BB	1060	**1060**	1060	1060	1060	1060
15	GA	683	**669**	669	669	669	669	60	BB	1067	**1067**	1067	1104	1102	1072
16	GA	982	**931**	931	931	931	931	61	GA	2175	**2327**	2382	2447	2452	2493
17	PR	1122	**1122**	1128	1128	1128	1128	62	GA	2962	**3272**	inf	3390	3451	3335
18	TS	978	**965**	965	998	1107	969	63	BB	inf	inf	inf	inf	inf	inf
19	GA	1084	inf	inf	inf	inf	inf	64	GA	2160	**2276**	2311	2445	2416	2448
20	GA	1027	inf	**1058**	inf	inf	1174	65	BB	inf	inf	inf	inf	inf	inf
21	BB	717	**717**	717	717	717	717	66	GA	3167	inf	inf	inf	inf	inf
22	BB	983	**983**	983	983	983	983	67	GA	2905	inf	inf	inf	inf	inf
23	AM	848	**848**	848	848	848	848	68	GA	2337	**2565**	inf	inf	inf	inf
24	BB	1107	**1107**	1107	1107	1107	1107	69	GA	2459	**2487**	2540	2593	2585	2599
25	BB	1027	**1027**	1027	1027	1027	1027	70	GA	2123	inf	inf	inf	inf	inf
26	BB	804	**804**	804	804	804	804	71	GA	1343	1289	**1278**	1299	1302	1313
27	BB	749	**749**	749	749	749	749	72	FB	1437	1431	**1429**	1431	1433	1433
28	BB	913	**913**	913	913	913	913	73	GA	1925	inf	**1970**	inf	2054	2053
29	BB	893	**893**	893	893	893	893	74	GA	2459	inf	2500	inf	**2489**	2501
30	BB	792	**792**	792	792	792	792	75	BB	976	**976**	977	976	976	976
31	BB	inf	inf	inf	inf	inf	inf	76	GA	2077	2100	2100	**2099**	2099	2108
32	BB	inf	inf	inf	inf	inf	inf	77	FB	1047	**1043**	1045	1049	1049	1049
33	GA	2343	inf	inf	inf	inf	inf	78	FB	2011	**2048**	2088	inf	inf	2098
34	BB	inf	inf	inf	inf	inf	inf	79	TS	1727	1752	1751	**1748**	1754	1749
35	BB	inf	inf	inf	inf	inf	inf	80	GA	1462	1539	**1464**	1539	1559	1593
36	GA	2211	**2316**	2355	2449	2447	2447	81	BB	1164	**1164**	1164	1164	1164	1164
37	GA	2318	inf	inf	inf	inf	inf	82	BB	1238	**1238**	1238	1238	1238	1238
38	TS	2575	inf	inf	inf	inf	inf	83	AM	1849	**1873**	1880	1877	1877	1876
39	BB	inf	inf	inf	inf	inf	inf	84	BB	936	**936**	937	936	938	936
40	GA	2628	inf	inf	inf	inf	inf	85	BB	1418	**1418**	1418	1418	1418	1418
41	TS	1243	1218	**1214**	1231	1233	1227	86	BB	1420	**1420**	1420	1420	1420	1420
42	FB	1038	1069	**1051**	1064	1268	1051	87	AM	1276	1274	**1273**	1276	1274	inf
43	TS	1354	**1332**	1333	1338	1366	1343	88	BB	1300	**1300**	1300	1300	1300	1300
44	AM	801	**801**	801	801	801	801	89	BB	1419	**1419**	1419	1419	1419	1419
45	BB	1088	**1088**	1088	1088	1088	1088	90	BB	1062	**1062**	1062	1062	1062	1062

Instances are from package UBO500.