Scheduling environments and heuristics

1. Single machine scheduling

Although the simplest in structure, scheduling on a single machine is still NP hard under most conditions, depending on the criteria of evaluation. Single machine heuristics often form the basis for more complex procedures for deployment in other environments.

For all the described variations on a single machine scheduling, the following is presumed:

- all data on the jobs are known in advance (offline scheduling),
- scheduling is performed dynamically (during the system run),
- jobs are non-preemptive,
- the machine (tool) is continuously available.

These assumptions are in accordance with the terms and conditions in the majority of contributions for this problem in the literature, and are retained here for the sake of easier comparison. On the other hand, this does not mean that the described methods cannot be used in systems in which some of these assumptions are not met (e.g. information about jobs can always be changed during the operation of the system).

1.1 Static scheduling environment

Static environment presumes the availability of jobs from the start of the system, i.e. ready time of each job is zero. Existence of recedence constraints or setup times is handled later in the text.

Scheduling heuristics in static environment

For each of the rules a priority function is used to determine job priority, and the best priority is the one with the highest value.

- **WSPT (weighted shortest processing time):**
  \[ \pi_j = \frac{w_j}{p_j}. \] (0.1)
- **EDD (earliest due date):**
  \[ \pi_j = \frac{1}{d_j}. \] (0.2)
- **Montagne [Dim 01, Mor 93] (short: MON):**
  \[ \pi_j = \left( \frac{w_j}{p_j} \right) \left( 1 - \frac{d_j}{\sum_{i=1}^{n} p_i} \right), \] (0.3)
  where \( n \) is the number of jobs.
- **Rachamadugu & Morton rule, (short: RM, ATC) [Moh 83] [Mor 93]:**
  \[ \pi_j = \left( \frac{w_j}{p_j} \right) \left[ \exp \left( \frac{-\left( d_j - p_j - time \right)^+}{k \cdot p_{AV}} \right) \right], \] (0.4)
  where \( p_{AV} \) is average duration of all unscheduled jobs, \( time \) is the current time, and operator \(^+\) is defined as:
\[ x^* = \max \{ x, 0 \} \tag{0.5} \]

The parameter \( k \) may depend on the particular version of the problem and the authors suggest a value of 2 for problems on a single machine.

In addition to these rules, for the criterion of weighted sum of tardy jobs a heuristic is implemented from [Mor 93] based on Hodgson's algorithm (which gives an optimal solution to the unweighted problem). The procedure is shown as Algorithm 1, and is in results marked with 'HODG'. It should be noted that activities that are 'ejected' from the list are actually transferred to the end of the timetable, and their order is irrelevant (for a given criterion) because they are all late.

\begin{verbatim}
sort jobs in a list using EDD algorithm;
calculate the completion time of all jobs in the current sequence;
while (there is a delay in operations in the list)
{k = the first late job;
find a job \( r \) for which \( \frac{w_r}{p_r} = \min \{ \frac{w_i}{p_i} \}, i=1..k \);
delete job \( r \) from the list;
calculate new completion times of all tasks;
} schedule jobs by the order in the list;
schedule deleted tasks in any order;\
\end{verbatim}

Algorithm 1. Weighted sum of tardy jobs heuristic (HODG)

1.2 Dynamic scheduling environment

Dynamic environment introduces ready times of jobs, which represent the time when the machine can start processing a job. The most common evaluation criteria in this environment are also weighted tardiness and weighted number of tardy jobs.

Scheduling heuristics in dynamic environment

In a dynamic environment, where jobs arrive over time, the existing scheduling heuristics that presume all jobs are available are modified according to [Mor 93] so that the processing time of a job is increased by the amount of time till the job becomes ready ('arrival'), defined as

\[ ar = \max \{ r_j - \text{time}, 0 \} \tag{0.6} \]

The heuristics modified in this way for dynamic problems are denoted with a suffix 'd', e.g. WSPT_d. The question remains as to which jobs to include when calculating the priority function? It can be shown that, for any regular scheduling criteria [Mor 93], a job should not be scheduled if the arrival for that job is longer than the processing time of the shortest of all currently available unscheduled jobs. In other words, we may only consider jobs \( j \) for which

\[ |r_j - \text{time}| \leq \min \{ p_i \}, \forall i : r_i \leq \text{time} \tag{0.7} \]

When testing in dynamic environment, we use the following meta-algorithm with an arbitrary priority function:

\begin{verbatim}
while there are unscheduled jobs do
{ wait until the machine and at least one job are ready;
P_{MIN} = processing time of the shortest available job;
\end{verbatim}
calculate priorities of all jobs \( j \) with \( |r_j - \text{time}| < p_{\text{MIN}} \); schedule job with best priority;

### Algorithm 2: Scheduling in a dynamic environment

The observed scheduling rules are listed below:

- **WSPT\_d:**
  \[
  \pi_j = \frac{w_j}{(p_j + ar_j)},
  \]
  \( 0.8 \)

- **EDD\_d**, priority equal to \( 0.2 \).

- **Montagne rule (MON\_d):**
  \[
  \pi_j = \left( w_j / (p_j + ar_j) \right) \cdot \left( 1 - \frac{d_j}{\sum_{i=1}^{n} p_i} \right).
  \]
  \( 0.9 \)

- **Rachamadugu & Morton (RM\_d):**
  \[
  \pi_j = \left( w_j / (p_j + ar_j) \right) \left[ \exp \left( -\frac{(d_j - (p_j + ar_j) - \text{time})^+}{k \cdot p_{AV}} \right) \right].
  \]
  \( 0.10 \)

- **X-dispatch dotlleneck dynamics heuristic [Mor 93] (short: XD):**
  \[
  \pi_j = \left( w_j / p_j \right) \left[ \exp \left( -\frac{(d_j - p_j - \text{time})^+}{k \cdot p_{AV}} \right) \right] \cdot \left( 1 - B \frac{(r_j - \text{time})^+}{p_{AV}} \right),
  \]
  \( 0.11 \)

where \( p_{AV} \) is average duration of unscheduled jobs, and \( B \) is defined as \( B = 1.3 + \rho \). Parameter \( \rho \) is average load, and for the purposes of testing its value was varied in \( [0.5, 1] \) with increments of 0.1.

### 1.3 Precedence constraints

The following conditions must be met before a job can be started:
- job \( i \) must be ready, \( r_i \leq \text{time} \) and
- all directly preceding jobs must be finished.

### Scheduling heuristics with precedence constraints

The existing heuristics can also be used with precedence constraints, as long as the scheduler considers only those jobs whose predecessors have finished. However, this would yield poor results since precedence information is not included in priority calculation. In this scheduling environment we include additional benchmark heuristics:

- Highest level heuristic (HL) chooses the job with the highest level in the precedence graph, among the available jobs. If we define the path length as the sum of processing times of all jobs (nodes) in a path in the graph, then the level of a job is defined as the length of the longest path from that job to any job with no successors (i.e. any node with no child nodes).
• LNS (engl. largest number of successors) pravilo prvo raspoređuje posao koji ima najviše neposrednih sljedbenika (funkcija prioritetu jednaka je broju podčvorova zadanog čvora).

A more complex Sidney algoristic (SIDNEY) is implemented as given in [Dha 78]. The word ‘algoristic’ is used because it can be shown that this procedure is optimal for assembly tree and near optimal for general tree precedence graphs with weighted tardiness and weighted flowtime as objectives.

This procedure uses the concepts of initial set, simple initial set and the associated values that are defined as follows:

• initial set is the set of jobs for which all their predecessors are also members of this set;
• simple initial set of a job contains the job and all its predecessors.

For any set of jobs aggregate values can be defined as the sum of the corresponding values of all elements of this set. For example, aggregate duration of a set of jobs is equal to the sum of all job durations in the set:

\[ p_S = \sum_{j \in S} p_j. \]  

Sidney heuristic is listed as Algorithm 3.

```
S = set of all unscheduled jobs;
while (S not empty) {
    define simple initial sets for all jobs in S;
    A = simple init. set with greatest aggregate \( w_A/P_A \);
    if (A has only one element)
        { schedule job from A;
           update S;
        }
    inače
        S = A bez posljednjeg elementa;
}
```

Algorithm 3. Sidney heuristic

In the described algorithm, "the last element" of the simple initial set is always the job that we begin to build a set with, i.e. the one that has no successors. It can be proven [Dha 78] that this algorithm gives optimal solution to weighted tardiness criteria for systems in which constraints are presented with a tree where each node has a maximum of one immediate successor (assembly tree, intree).

### 1.4 Sequence dependant setup times

When scheduling a series of jobs on a machine, it is understood that the duration of the activity involves the time required for the adjustment before starting the job. If the duration of each job adjustment is the same regardless of what was previously completed on the machine, this duration may not be separated from the total duration of the job. An example of this can be a context switch in computer memory before the start of the next process, which is usually independent of the type of the previous process. It is possible, however, that the duration of the adjustment depends on what job was previously done, in which case there are sequence dependent setup times.
Scheduling heuristics with setup times

Almost any heuristic may be adjusted to include sequence dependent setup time. Job priority obtained with the original function is decreased by a certain value that measures the additional cost brought by setup time for that job. If the original priority value is denoted with $\pi_j$, then the priority with setup times is given with

$$\pi_{sj} = \pi_j - \frac{s_{lj}}{p_{AV} \cdot p_j} \quad (0.13)$$

where $l$ is the last processed job, $s_{lj}$ setup time between job $l$ and job $j$ and $p_{AV}$ the average processing time of all unscheduled jobs. The heuristics adapted in this way are denoted with suffix ‘s’ (e.g. RM_s). We include the following heuristics:

- **WSPT_s**:

  $$\pi_j = \frac{w_i}{p_j} - \frac{s_{lj}}{p_{AV} \cdot p_j} \quad (0.14)$$

- **MON_s**:

  $$\pi_j = \left( \frac{w_j}{p_j} \right) \cdot \left( 1 - \frac{d_j}{\sum_{i=1}^{n} p_i} \right) - \frac{s_{lj}}{p_{AV} \cdot p_j} \quad (0.15)$$

- **RM_s**:

  $$\pi_j = \left( \frac{w_j}{p_j} \right) \left[ \exp \left( \frac{-\left( d_j - p_j - time \right)^+}{k_1 p_{AV}} \right) \right] - \frac{s_{lj}}{p_{AV} \cdot p_j} \quad (0.16)$$

- **ATCS (Apparent Tardiness Cost with Setups, [Lee 97])**:

  $$\pi_j = \left( \frac{w_j}{p_j} \right) \left[ \exp \left( \frac{-\left( d_j - p_j - time \right)^+}{k_1 \bar{p}} \right) \right] - \frac{s_{lj}}{k_2 \bar{s}} \quad (0.17)$$

where $\bar{p}$ is average duration of all jobs, $\bar{s}$ the average setup time for all pairs of jobs, and $k_1$ and $k_2$ the scaling parameters. The parameters $k_1$ and $k_2$ are determined prior to scheduling and are used to evaluate the degree of problem hardness. The authors propose the following way of calculating these parameters; first to estimate the value of the parameter $T$ (due date tightness) and $R$ (due date range):

$$R = \frac{d_{j_{MAX}} - d_{j_{MIN}}}{\hat{C}_{MAX}}, \quad T = 1 - \frac{d_{AVG}}{\hat{C}_{MAX}} \quad (0.18)$$

where $d_{j_{MAX}}$, $d_{j_{MIN}}$ and $d_{AVG}$ are greatest, smallest and average due date, and $\hat{C}_{MAX}$ is completion time estimate:

$$\hat{C}_{MAX} = \sum_{j=1}^{n} p_j + n \cdot \bar{s} \quad (0.19)$$
In addition, the ratio of setup time and duration of jobs is estimated:

\[ \eta = \frac{\bar{s}}{\bar{p}}, \]  

(0.20)

after which \( k_1 \) and \( k_2 \) are determined as:

\[ k_1 = \begin{cases} 4.5 + R, & \text{for } R \leq 0.5 \\ 6 - 2R, & \text{for } R > 0.5 \end{cases}, \]

(0.21)

\[ k_2 = \frac{T}{2\sqrt{\eta}}. \]

- ATCSR (Apparent Tardiness Cost with Setups and Ready times, [Pfu 08]):

\[ \pi_j = \frac{w_j}{p_j} \exp \left( -\frac{\left( d_j - p_j - \max \left( r_j - \text{time} \right) \right)^+}{k_1\bar{p}} - \frac{s_j}{k_2\bar{s}} - \frac{\left( r_j - \text{time} \right)^+}{k_3\bar{p}} \right), \]  

(0.22)

where \( \bar{p} \) is average duration of all jobs, \( \bar{s} \) the average setup time for all pairs of jobs, and \( k_1, k_2 \) and \( k_3 \) the scaling parameters. In our experiments the scaling parameters for this method were varied (grid approach) and the best obtained schedule is used as a result. The parameters assumed the following values: \( k_1 \in \{0.2, 0.6, 0.8, 1.4, 2, 2.8, 4, 5.2, 6.4, 7.2\} \), \( k_2 \in \{0.1, 0.3, 0.7, 1.3, 1.7, 2.1\} \), and \( k_3 \in \{0.0025, 0.004, 0.025, 0.05, 0.25, 0.4, 0.6, 1.2\} \).

2. **Uniform parallel machines scheduling**

For all the included variations of uniform parallel machines, the following is presumed:

- all data on the jobs are known in advance (offline scheduling),
- scheduling is done dynamically (during the system run)
- jobs are non-preemptive,
- machines are continuously available from the beginning.

Although a dynamic availability of machines is not considered, all the rules can be adapted very easily to this change, since in the process of scheduling we are already waiting for a machine to become available before making a decision.

Scheduling on parallel identical or uniform machines with dynamic availability and with any regular criterion can always be reduced to a single machine problem. The optimal solution can be obtained as follows: sort all tasks in the 'proper' order and assign each job on a machine that can first finish the job. It is advisable, however, to provide to the scheduling rule further information about the properties of the environment.

Scheduling rules defined in the previous section can be used in unmodified form. If permitted by the dynamic availability of jobs, job duration is also increased by the amount of time until the arrival of the job as described in section 1.2. In case of setup times certain rule can be adjusted in the manner described in section 1.4 (0.13). This way, in this chapter WSPT rule, MON, ATCS and the EDD are used. In addition is included the LPT (engl. longest processing time) rule whose priority is defined as

\[ \pi_j = p_j. \]  

(0.23)
Additional included heuristics are the following:

- **mRM (modified R&M):**
  \[
  \pi_j = \frac{w_j}{(p_j/m)} \left[ \exp \left( \frac{-(d_j - p_j/s_k - time)^+}{(p_{AV}/m)} \right) \right] ; \quad (0.24)
  \]

- **mXD (modified X-dispatch):**
  \[
  \pi_j = \frac{w_j}{(p_j/m)} \left[ \exp \left( \frac{-(d_j - (p_j/s_k) - time)^+}{(p_{AV}/m)} \right) \right] \left[ 1 - B \frac{(r_j - time)^+}{(p_{AV}/m)} \right] , \quad (0.25)
  \]

Since the definition of the priorities of these rules includes the effective number of machines, the adjustment for setup times is defined by the formula

\[
\pi_{ij} = \pi_j - \frac{s_{ij}}{(p_{AV}/m)(p_j/s_k)} , \quad (0.26)
\]

where \( l \) is the previous job (on machine \( k \)), \( s_k \) is machine speed and \( s_{ij} \) is the setup time (machine independent).

### 3. Unrelated parallel machines scheduling

Okolina nesrodnih strojeva u velikoj se mjeri razlikuje od okruženja jednolikih strojeva razmatranog u prethodnom poglavlju. Iako se svaki posao i dalje može izvesti na bilo kojem stroju, ovaj put je pridruživanje posla određenom stroju od velike važnosti jer su trajanja obrade poslova na različitim strojevima potpuno neovisna. Potrebno je stoga algoritmu raspoređivanja omogućiti uvid u prikladnost pojedinoga posla pojedinome stroju i na učinkovit način iskoristiti tu informaciju.

U prethodnim poglavljima korišten je model predodređenog raspoređivanja, u kojemu su svi podaci o poslovima i njihovim dolascima bili dostupni. U ovom će se poglavlju, ponajprije zbog načina rada postupaka za usporedbu (sljedeći odjeljak), rabiti model raspoređivanja na zahtjev. U takvim uvjetima raspoređivanja podaci o poslovima dostupni su tek po dolasku posla u sustav, što se u većini primjera poklapa sa vremenom pripravnosti posla. Uz ovu razliku, vrijede sljedeće pretpostavke:

- podaci o poslovima nisu unaprijed poznati (raspoređivanje na zahtjev), što povlači dinamički način izrade rasporeda,
- poslovi su neprekidni,
- strojevi su neprekidno raspoloživi od početka rada sustava.

#### 3.1 Postupci raspoređivanja na nesrodnim strojevima

Prilikom raspoređivanja na nesrodnim strojevima, dolaze do izražaja dvije faze raspoređivanja poslova: pridruživanje, pri kojemu se poslovi dodjeljuju odgovarajućim strojevima, i uređivanje, pri čemu se određuje redoslijed eventualnog većeg broja poslova na jednom stroju. Neki postupci raspoređivanja ove dvije faze obavljaju odvojeno, a kod nekih je jedna sadržana u drugoj (npr. uređivanje se obavlja redom kojim su poslovi pridruženi stroju). Također je dvije različite metode pridruživanja i uređivanja moguće obuhvati u jedinstveni postupak raspoređivanja.
Algoritmi raspoređivanja u dinamičkim uvjetima raspoređivanja na zahtjev mogu se razlikovati po tome u kojem trenutku je neki posao raspoređen na odgovarajući stroj. Jedna skupina algoritama raspoređuje pristigli posao na neki od strojeva odmah po njegovom dolasku, čime se zapravo obavlja samo pridruživanje poslova. Uređivanje poslova na stroju podrazumijeva redoslijed izvođenja jednak redu prispjevaca, tj. uređivanje po FCFS (engl. *first come first served*) principu. Druga skupina algoritama ne raspoređuje svaki posao zasebno odmah po njegovu dolasku, već se u predodređenim vremenskim intervalima zajednički raspoređuju svi do tada pristigli a još nepokrenuti poslovi (engl. *batch mode*). Skup svih poslova (zadataka) koji su trenutno u sustavu, a izvođenje im nije pokrenuto, naziva se još i *meta-zadatkom*. Prilikom raspoređivanja, obavlja se i pridruživanje i uređivanje svih poslova u meta-zadatku. U većini primjena algoritmi iz druge skupine postižu bolje rezultate, pa će se u ovom radu promatrati postupci iz te skupine.

Za okruženje nesrodnih strojeva postoji niz heuristika raspoređivanja [Mar 04, Dav 81], no najpoznatije metode su min-min i max-min heuristike. Ova dva algoritma (pogotovo min-min) smatraju se ponajboljim heurističkim algoritmicima za veliki broj problema raspoređivanja [He 03, Mar 04]. Algoritmi raspoređivanja koji će kasnije dobiti naziv min-min i max-min predloženi su još u radu [Iba 77] uz nekoliko drugih postupaka. Prije opisa algoritma, potrebno je uvesti sljedeće oznake:

- \( p_j \) je trajanje izvođenja posla \( j \) na stroju \( m_i \); sva trajanja izvođenja su unaprijed poznata sa dovoljnom preciznošću;
- \( t_i \) je vrijeme pripravnosti stroja \( m_i \) (s obzirom na poslove koji se na njemu trenutno izvode);
- \( C_j \) je vrijeme završetka izvođenja posla \( j \) na stroju \( m_i \); budući su vremena završetka vezana samo uz poslove, za određivanje kriterija koristi se oznaka \( C_j \) kao vrijeme završetka posla \( j \);
- \( J^\text{meta} \) je skup svih raspoloživih poslova koji još nisu pokrenuti; neki od tih poslova su možda dodijeljeni nekome stroju, ali njihovo izvođenje još nije započelo pa su podložni novom pridruživanju.

Min-min postupak je prikazan kao algoritam 4. U svakom koraku algoritam računa očekivano vrijeme završetka izvođenja poslova na svakom od strojeva, uzevši u obzir vrijeme pripravnosti stroja. Potom se za svaki posao pronašta stroj koji daje najmanje očekivano vrijeme završetka. Postupak tada raspoređuje posao koji postiže najbliže vrijeme završetka (od svih poslova) na stroj koji tom poslu daje to vrijeme završetka. Max-min algoritam također za svaki posao pronašta stroj koji daje najmanje vrijeme završetka, no raspoređuje se onaj posao koji ima najveće očekivano (minimalno) vrijeme završetka među svim poslovima. U primjeni se mogu naći i postupci zasnovani na min-min algoritmu, koji uz neke prilagodbe danom okruženju najčešće daju nešto bolje rezultate raspoređivanja [He 03].

**za** (sve poslove iz \( J^\text{meta} \))

**za** (sve strojeve)

\[ C_j = t_i + p_j \]

**dok** (postoje neraspoređeni poslovi u \( J^\text{meta} \))

{ za (svaki zadatak iz \( J^\text{meta} \))

pronaći najmanji \( C_j \) te stroj \( m_i \) na kojemu se postiže najranije vrijeme završetka;

naći posao \( j \) sa najmanjim vremenom završetka;

dodijeliti posao \( j \) stroju \( l \) koji daje najranije vrijeme završetka;
4. Job shop scheduling

This environment is represented by scheduling jobs, each of which consists of a number of operations. Operations are carried out in predefined order on specific machines. Scheduling is typically performed as offline scheduling, where all parameters are known in advance (number of operations and their duration on machines). In addition, it is presumed operations are non-preemptive and machines are available in the entire duration of the system run.

4.1 Scheduling heuristics for job shop environment

Algorithms for this environment are given much attention, so for this problem there are many ways of scheduling with different techniques (evolutionary methods, expert systems, fuzzy decision-making, etc.) [Kas 99, Jon 98, Cic 01]. The primary focus of this study were procedures that can be applied in the form of scheduling rules and in dynamic conditions. A considerable number of examples of simple scheduling rules for this problem can be found in [Cha 96]. The general procedure of applying the rules is shown as algorithm 5.

All scheduling rules are provided with information about jobs, their operations and condition of the machines, and the following notation is used:

- $p_{ij}$ – duration of single operation of job $j$ on machine $i$;
- $w_j$ – job weight;
- $d_j$ – job due date;
- $twk_j$ – (total work) total duration of all job operations;
- $twkr_j$ – (total work remaining) total duration of all unstarted operations of a job.

while (there are unscheduled operations)
{
  wait until a machine for which there are operations becomes available;
  determine the priorities of all operations waiting on the machine;
schedules the operation with the highest priority;
determine the next operation of the job;
update availability time of machine and the following job operation;

Algorithm 5. Scheduling in job shop environment

The following scheduling rules are included:

- **WSPT:**

\[
\pi_j = \frac{w_j}{p_{ij}}. 
\]  \hspace{1cm} (0.27)

- **WSPT/TWKR:**

\[
\pi_j = \frac{w_j \cdot twkr_j}{p_{ij}}. 
\]  \hspace{1cm} (0.28)

- **WTWKR** (weighted total work remaining):

\[
\pi_j = \frac{w_j}{twkr_j}. 
\]  \hspace{1cm} (0.29)

- **SLACK/TWKR pravilo** (slack per remaining process time):

\[
\pi_j = \frac{w_j \cdot twkr_j}{(d_j - twkr_j)}. 
\]  \hspace{1cm} (0.30)

- **EDD:**

\[
\pi_j = \frac{1}{d_j}. 
\]  \hspace{1cm} (0.31)

- **COVERT** (cost over time) [Mor 93, p. 340]:

\[
\pi_j = \frac{w_j}{p_{ij}} \left[ 1 - \left( \frac{d_j - ELT_{ij} - \text{time}}{h \cdot ELT_{ij}} \right)^+ \right], 
\]  \hspace{1cm} (0.32)

where \( h \) is heuristic parameter, \( ELT_{ij} \) (estimated lead time) is an estimation of the time the job \( j \) will spend in the system after current operation finishes (\( i \)). This value is in most cases defined as \( ELT_{ij} = k \cdot twkr_j \). The usual value for \( k \) is 3, and for parameter \( h \) the authors suggest the value of 0.5 [Mor 93].

- **RM rule for job shop** [Mor 93, p. 340]:

\[
\pi_j = \frac{w_j}{p_{ij}} \cdot \exp \left[ \frac{(d_j - ELT_{ij} - \text{time})^+}{h \cdot \bar{p}_i} \right], 
\]  \hspace{1cm} (0.33)

where \( \bar{p}_i \) is average duration of all operations on given machine. Parameter \( h \) was here empirically determined as 10, considering the results on used test cases, and \( ELT_{ij} \) is determined in the same way as in COVERT rule.
Although the jobs are considered to be available from time zero, scheduling on a given machine is inherently dynamic because an operation may only be ready at some time in the future (after the completion of the job’s previous operation). We must therefore define the priority calculation for the operations available in the future, as in the single machine dynamic problem. This can be resolved in the following ways:

1. no inserted idleness - we only consider those operations which are immediately available;
2. inserted idleness - waiting for an operation is allowed and waiting time is added to operation's processing time in priority calculation (as was the case in single machine dynamic environment);
3. inserted idleness with arbitrary priority - waiting is allowed but the priority function must be defined so it takes waiting time into account.

For test cases used in this work, the scheduling heuristics used for comparison purposes exhibit better performance with the first method, so the first method is used when comparing efficiency with genetic programming evolved algorithm. The genetic programming, on the other hand, is coupled with the third method, as it has the ability to learn and make use of waiting time information on itself.
References


Evolution of scheduling heuristics with GP

[Cic 01] Vincent A. Cicirello, Stephen F. Smith, Ant Colony Control for Autonomous Decentralized Shop Floor Routing, Fifth International Symposium on Autonomous Decentralized Systems March 26 - 28, 2001 Dallas, Texas p. 383,
http://csdl.computer.org/comp/proceedings/isads/2001/1065/00/10650383abs.htm

http://www.ri.cmu.edu/pubs/pub_3789.html

http://www.cs.drexel.edu/~cicirello/benchmarks.html


[Cra 85] Michael Lynn Cramer, A Representation for the Adaptive Generation of Simple Sequential Programs, International Conference on Genetic Algorithms and their Applications [ICGA85], CMU, Pittsburgh,
http://www.sover.net/~nichael/nlc-publications/icga85/index.html

http://portal.acm.org/citation.cfm?doid=322227.322284


http://www.sciencedirect.com/


http://citeseer.ist.psu.edu/eiben00parameter.html


Evolution of scheduling heuristics with GP  
http://gp.zemris.fer.hr/scheduling


[Koz 05] John R. Koza, Genetically Breeding Populations of Computer Programs to Solve Problems in Artificial Intelligence , Proceedings of the Second International Conference on Tools for AI, Herndon,
Evolution of scheduling heuristics with GP  http://gp.zemris.fer.hr/scheduling

Virginia, USA, 1990
http://citeseer.ist.psu.edu/koza90genetically.html

http://www.genetic-programming.com/jkpubs72to93.html


ftp://cs.ucl.ac.uk/genetic/papers/wbl_ppsn2000.ps.gz


http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html


[Lek 01] Lekin®, Flexible Job Shop Scheduling System
http://www.stern.nyu.edu/om/software/lekin/


[Lop 01] A. Lopez-Ortiz, Computational Theory FAQ,
http://db.uwaterloo.ca/~alopez-o/comp-faq/faq.html


http://www.math.tu-berlin.de/~nmegow/muv05sos.pdf


[Mon 01] Patrick Monsieurs, Eddy Flerackers, Detecting and Removing Inactive Code in Genetic Programs, 2001,
Evolution of scheduling heuristics with GP


http://citeseer.ist.psu.edu/276159.html


http://staff.aist.go.jp/k.miyashita/publications/PRICAI2000.ps

http://staff.aist.go.jp/k.miyashita/publications/ccc.ps

http://www.dcs.st-and.ac.uk/~norman/Pubs/cache.ps.gz

[Pen 00] Carlos Andrés Peña-Reyes, Moshe Sipper, *Evolutionary computation in medicine: an overview*, Artificial Intelligence in Medicine Volume 19, Issue 1, 1 May 2000, Pages 1-23,
http://www.sciedirect.com/


http://citeseer.ist.psu.edu/328504.html


http://www.sciedirect.com/

Evolution of scheduling heuristics with GP  


http://www.cs.uidaho.edu/~tsoule/research/the3.ps


http://ina.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html

[Tal 03] W. A. Talbott, Automatic Creation of Team-Control Plans Using an Assignment Branch in Genetic Programming, Genetic Algorithms and Genetic Programming at Stanford 2003, 


