# 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*):

$$\pi_j = w_j / p_j \,. \tag{0.1}$$

- EDD (*earliest due date*):

$$\pi_j = 1/d_j \,. \tag{0.2}$$

- Montagne [Dim 01, Mor 93] (short: MON):

$$\pi_j = \left(w_j / p_j\right) \cdot \left(1 - \frac{d_j}{\sum_{i=1}^{n} p_i}\right), \tag{0.3}$$

  where $n$ is the number of jobs.

- Rachamadugu & Morton rule, (short: RM, ATC) [Moh 83] [Mor 93]:

$$\pi_j = \left(w_j / p_j\right) \left[\exp\left(\frac{-\left(d_j - p_j - time\right)^+}{k \cdot p_{AV}}\right)\right], \tag{0.4}$$

  where $p_{AV}$ is average duration of al 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.

```
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 w_r / p_r = min{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;
```

*Algoritam 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

$$ar = \max\{r_j - time, 0\}. \tag{0.6}$$

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

$$\left|r_j - time\right| \le \min_i\{p_i\}, \forall i : r_i \le time \tag{0.7}$$

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

```
while there are unscheduled jobs do
{     wait until the machine and at least one job are ready;
      p_jMIN = processing time of the shortest available job;
```

```
    calculate priorities of all jobs j with |r_j − time| < p_jMIN ;
    schedule job with best priority;
}
```

*Algoritam 2.Scheduling in a dynamic environment*

The observed scheduling rules are listed below:

- WSPT_d:

$$\pi_j = w_j / \left( p_j + ar_j \right). \tag{0.8}$$

- EDD_d, priority equal to (0.2).
- Montagne rule (MON_d):

$$\pi_j = \left( w_j / \left( p_j + ar_j \right) \right) \cdot \left( 1 - \frac{d_j}{\sum_{i=1}^{n} p_i} \right). \tag{0.9}$$

- Rachamadugu & Morton (RM_d):

$$\pi_j = \left( w_j / \left( p_j + ar_j \right) \right) \left[ \exp\left( \frac{-\left( d_j - \left( p_j + ar_j \right) - time \right)^+}{k \cdot p_{AV}} \right) \right]. \tag{0.10}$$

- *X-dispatch dottleneck dynamics heuristic* [Mor 93] (short: XD):

$$\pi_j = \left( w_j / p_j \right) \left[ \exp\left( \frac{-\left( d_j - p_j - time \right)^+}{k \cdot p_{AV}} \right) \right] \cdot \left( 1 - B \frac{\left( r_j - time \right)^+}{p_{AV}} \right), \tag{0.11}$$

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, $r_i \le time$ 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 prioriteta 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 contans 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:

$$p_S = \sum_{j \in S} p_j \,. \tag{0.12}$$

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;
}
```

*Algoritam 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 dependant 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

$$\pi_{lj} = \pi_j - \frac{s_{lj}}{p_{AV} \cdot p_j} \tag{0.13}$$

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:

$$\pi_j = \frac{w_j}{p_j} - \frac{s_{lj}}{p_{AV} \cdot p_j}, \tag{0.14}$$

- MON_s:

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

- RM_s:

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

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

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

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):

$$R = \frac{d_{jMAX} - d_{jMIN}}{\hat{C}_{MAX}}, \quad T = 1 - \frac{d_{AVG}}{\hat{C}_{MAX}}, \tag{0.18}$$

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

$$\hat{C}_{MAX} = \sum_{j=1}^{n} p_j + n \cdot \bar{s}\ . \tag{0.19}$$

In addition, the ratio of setup time and duration of jobs is estimated:

$$\eta = \frac{\overline{s}}{\overline{p}}, \tag{0.20}$$

after which $k_1$ and $k_2$ are determined as:

$$\begin{aligned} k_1 &= \begin{cases} 4.5 + R, & for\ R \le 0.5 \\ 6 - 2R, & for\ R > 0.5 \end{cases}, \\ k_2 &= \frac{T}{2\sqrt{\eta}}. \end{aligned} \tag{0.21}$$

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

$$\pi_j = \frac{w_j}{p_j} \exp\left( - \frac{\left(d_j - p_j - \max\left(r_j - time\right)\right)^+}{k_1 \overline{p}} - \frac{s_{lj}}{k_2 \overline{s}} - \frac{\left(r_j - time\right)^+}{k_3 \overline{p}} \right), \tag{0.22}$$

where $\overline{p}$ is average duration of all jobs, $\overline{s}$ 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: $k_1 \in \{0.2, 0.6, 0.8, 1.4, 2, 2.8, 4, 5.2, 6.4, 7.2\}$, $k_2 \in \{0.1, 0.3, 0.7, 1.3, 1.7, 2.1\}$, $k_3 \in \{0.0025, 0.004, 0.025, 0.05, 0.25, 0.4, 0.6, 1.2\}$.

## 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

$$\pi_j = p_j. \tag{0.23}$$

Additional included heuristics are the following:

- mRM (*modified R&M*):

$$\pi_j = \frac{w_j}{\left(p_j/\hat{m}\right)}\left[\exp\left(\frac{-\left(d_j - p_j/s_k - time\right)^+}{\left(p_{AV}/\hat{m}\right)}\right)\right]; \tag{0.24}$$

- mXD (*modified X-dispatch*):

$$\pi_j = \frac{w_j}{\left(p_j/\hat{m}\right)}\left[\exp\left(\frac{-\left(d_j - \left(p_j/s_k\right) - time\right)^+}{\left(p_{AV}/\hat{m}\right)}\right)\right]\cdot\left(1 - B\frac{\left(r_j - time\right)^+}{\left(p_{AV}/\hat{m}\right)}\right), \tag{0.25}$$

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

$$\pi_{lj} = \pi_j - \frac{s_{lj}}{\left(p_{AV}/\hat{m}\right)\cdot\left(p_j/s_k\right)}, \tag{0.26}$$

where $l$ is the previous job (on machine $k$), $s_k$ is machine speed and $s_{lj}$ 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 neprekidivi,
- 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 obuhvatiti 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 prispijeća, 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 algoritmima 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:

- $p_{ij}$ je trajanje izvođenja posla $j$ na stroju $m_i$; sva trajanja izvođenja su unaprijed poznata sa dovoljnom preciznošću;
- $t_{ri}$ je vrijeme pripravnosti stroja $m_i$ (s obzirom na poslove koji se na njemu trenutno izvode);
- $C_{ij}$ 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 $j$;
- $J^{meta}$ 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 pronalazi 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 pronalazi 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^meta)
      za (sve strojeve)
          C_ij = t_ri + p_ij ;
dok (postoje neraspoređeni poslovi u J^meta)
{     za (svaki zadatak iz J^meta)
          pronaći najmanji C_ij 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;
```

```
      ukloniti posao j iz J^meta ;
      obnoviti t_rl = t_rl + p_lj ;
      obnoviti C_lj za sve preostale poslove iz J^meta ;
}
```

*Algoritam 4. Min-min postupak raspoređivanja*

Važan element uporabe opisanih postupaka raspoređivanja je i odabir trenutka raspoređivanja nagomilanih poslova. U stvarnim se uvjetima obično definira neki vremenski interval nakon kojega se algoritam ponovno primjenjuje. Određivanje veličine intervala u velikoj je mjeri ovisno o vrsti sustava u kojemu se raspoređivanje obavlja. U slučaju 'predugog' intervala može se dogoditi da neki pristigli posao, koji bi zbog 'dobrih' svojstava inače bio raspoređen na određeni stroj, ne bude raspoređen na taj stroj jer je na istom stroju već pokrenut posao koji je tom stroju dodijeljen u prethodnoj iteraciji raspoređivanja. Zbog toga se u projektu, u postupku simulacije, primjenjuje najbolji slučaj: algoritam se izvodi svaki puta kada novi posao dođe u sustav. Na taj način zajamčena je najbolja moguća učinkovitost algoritma.

# 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;
}
```

*Algoritam 5. Scheduling in job shop environment*

The following scheduling rules are included:

- WSPT:

$$\pi_j = \frac{w_j}{p_{ij}}. \tag{0.27}$$

- WSPT/TWKR:

$$\pi_j = \frac{w_j \cdot twkr_j}{p_{ij}}. \tag{0.28}$$

- WTWKR (*weighted total work remaining*):

$$\pi_j = \frac{w_j}{twkr_j}. \tag{0.29}$$

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

$$\pi_j = \frac{w_j \cdot twkr_j}{\left(d_j - twkr_j\right)}. \tag{0.30}$$

- EDD:

$$\pi_j = \frac{1}{d_j}. \tag{0.31}$$

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

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

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]:

$$\pi_j = \frac{w_j}{p_{ij}} \cdot \exp\left[\frac{\left(d_j - ELT_{ij} - time\right)^+}{h \cdot \overline{p}_i}\right], \tag{0.33}$$

where $\overline{p}_i$ 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


[Ada 02] T. P. Adams, *Creation of Simple, Deadline, and Priority Scheduling Algorithms using Genetic Programming,* Genetic Algorithms and Genetic Programming at Stanford 2002, http://www.genetic-programming.org/sp2002/Adams.pdf

[And 94] D. Andre, *Automatically Defined Features: The Simultaneous Evolution of 2-dimensional Featrue Detectors and an Algorithm for Using Them*, Advances in Genetic Programming, pp. 477-494, MIT Press, 1994.

[Atl 94] Bonnet L. Atlan, J.B. Polack, *Learning distributed reactive strategies by genetic programming for the general job shop problem,* Proceedings 7th annual Florida Artificial Intelligence Research Symposium, Pensacola, Florida, IEEE Press, 1994.

[Auy 03] Andy Auyeung, Iker Gondra, H. K. Dai, *Multi-heuristic list scheduling genetic algorithm for task scheduling,* Symposium on Applied Computing, Proceedings of the 2003 ACM symposium on Applied computing,Melbourne, Florida,Pages: 721 - 724, http://portal.acm.org/citation.cfm?doid=952532.952673

[Ban 02] W. Banzhaf, W. B. Langdon, *Some considerations on the reason for bloat,* Genetic Programming and Evolvable Machines, 3(1), pp. 81-91, March 2002.

[Ban 98] W. Banzhaf, P. Nordin, R. E. Kellet, F. D. Francone, *Genetic Programming – An Introduction: On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann, 1998.

[Bea 05] Open BEAGLE: a versatile EC framework, http://beagle.gel.ulaval.ca/

[Bea 90] J. E. Beasley, *OR-Library: Weighted tardiness*, 1990, http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html

[Bli 94] T. Blickle, L. Thiele, *Genetic Programming and Redundancy,* Proc. Genetic Algorithms within the Framework of Evolutionary Computation (Workshop at KI-94), Saarbrücken, Germany, 1994. , http://www.handshake.de/user/blickle/publications/GPandRedundancy.ps

[Bli 96] Tobias Blickle, *Evolving Compact Solutions in Genetic Programming: A Case Study,* Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation, LNCS, Vol. 1141, pp. 564-573, Springer-Verlag, 22-26 September 1996., http://www.blickle.handshake.de/publications/ppsn1.ps

[Bud 00] L. Budin, D. Jakobović, M. Golub, *Genetic Algorithms in Real-Time Imprecise Computing*, Journal of Computing and Information Technology CIT, Vol. 8, No. 3, September 2000, pp. 249-257.

[Bud 98] L. Budin, D. Jakobović, M. Golub, *Parallel Adaptive Genetic Algorithm*, Proc. Int. ICSC/IFAC Symposium on Neural Computation, NC'98, Vienna, September 23-25, 1998., pp. 157-163

[Bud 99] L. Budin, D. Jakobović, M. Golub, *Genetic Algorithms in Real-Time Imprecise Computing*, IEEE International Symposium on Industrial Electronics ISIE'99, Bled, 1999, Vol. 1, pp. 84-89.

[Bur 03] Edmund Burke, Steven Gustafson, Graham Kendall, *Ramped Half-n-Half Initialisation Bias in GP*, enetic and Evolutionary Computation -- GECCO-2003, LNCS, Vol. 2724, pp. 1800-1801, Springer-Verlag, 12-16 July 2003.

[Cha 04] Samarn Chantaravarapan, Jatinder N.D. Gupta, *Single Machine Group Scheduling with Setups to Minimize Total Tardiness,* 2004, http://www.pmcorp.com/PublishedPapers/Scheduling%20Publications/SingleMachineGroupSchedulingwithSetups.pdf

[Cha 96] Yih-Long Chang, Toshiyuki Sueyoshi, Robert Sullivan, *Ranking dispatching rules by data envelopment analysis in a job shop environment,* IIE Transactions, 28(8):631-642, 1996

[Che 99] V.H.L. Cheng, L.S. Crawford, P.K. Menon, *Air Traffic Control Using Genetic Search Techniques,* 1999 IEEE International Conference on Control Applications, August 22-27, Hawai'i, HA, http://www.optisyn.com/papers/1999/traffic_99.pdf

[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

[Cic 01a] Vincent Cicirello, Stephen Smith, *Randomizing Dispatch Scheduling Policies,* The 2001 AAAI Fall Symposium: Using Uncertainty Within Computation, November, 2001., http://www.ri.cmu.edu/pubs/pub_3789.html

[Cic 03] Vincent Cicirello, *Weighted Tardiness Scheduling with Sequence-Dependent Setups,* Technical report, The Robotics Institute, Carnegie Mellon Univeristy, 2003, http://www.cs.drexel.edu/~cicirello/benchmarks.html

[Cor 01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, 2nd ed., The MIT Press - McGraw-Hill, 2001.

[Cra 85] Nichael Lynn Cramer, *A Representation for the Adapative 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

[Dav 81] Ernest Davis, Jeffrey M. Jaffe, Algorithms for Scheduling Tasks on Unrelated Processors, Journal of the ACM, Volume 28 , Issue 4 (October 1981), pp. 721 – 736 http://portal.acm.org/citation.cfm?doid=322276.322284

[Dha 78] B. G. Dharan, T.E. Morton, Algoristics for Single Machine Sequencing with Precedence Constraints, Management Science 24, p. 1011-1020, 1978. http://www.ruf.rice.edu/~bala/files/dharan-morton-algoristics_for_sequencing-Mgt_Science_1978.pdf

[Dim 01] C. Dimopoulos, A. M. S. Zalzala, *Investigating the use of genetic programming for a classic one-machine scheduling problem,* Advances in Engineering Software, Volume 32, Issue 6 , June 2001, Pages 489-498, http://www.sciencedirect.com/

[Dim 99] Christos Dimopoulos, Ali M. S. Zalzala, *Evolving Scheduling Policies through a Genetic Programming Framework,* Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 2, p. 1231, Morgan Kaufmann, 13-17 July 1999.

[Dim 99a] Dimopoulos C., Zalzala A.M.S., *A genetic programming heuristic for the one-machine total tardiness problem,* Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on , Volume: 3 , 6-9 July 1999

[Eib 00] Ágoston E. Eiben, Robert Hinterding, Zbigniew Michalewicz, *Parameter Control in Evolutionary Algorithms,* IEEE Trans. on Evolutionary Computation, 2000. http://citeseer.ist.psu.edu/eiben00parameter.html

[Gag 02] C. Gagné, M. Parizeau, *Open BEAGLE: A New Versatile C++ Framework for Evolutionary Computation*, Late Breaking papers at the Genetic and Evolutionary Computation Conference (GECCO-2002), New York, USA, 9-13 July 2002

[Gag 03] Christian Gagné, Marc Parizeau, Marc Dubreuil, *Distributed BEAGLE: An Environment for Parallel and Distributed Evolutionary Computations,* Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS) 2003, http://vision.gel.ulaval.ca/en/publications/Id_439/PublDetails.php

[Gat 97] C. Gathercole, P. Ross, *Tackling the Boolean Even N Parity Problem with Genetic Programming and Limited Error Fitness*, Genetic Programming 1997: Proceedings of the 2nd Annual Conference, pp. 119-127, San Francisco, 1997

[Gib 02] K. A. Gibbs, *Implementation and Evaluation of a Novel Branch Construct for Genetic Programming,* Genetic Algorithms and Genetic Programming at Stanford 2002, http://www.genetic-programming.org/sp2002/Gibbs.pdf

[Gol 00] M. Golub, D. Jakobović, *A New Model of Global Parallel Genetic Algorithm*, Proc. 22th Int. Conference ITI'00, Pula, June 13-16, 2000

[Gol 01] M. Golub, *Poboljšavanje djelotvornosti paralelnih genetskih algoritama*, doktorska disertacija, Fakulet elektrotehnike i računarstva, 2001.

[Gol 01a] M. Golub, D. Jakobović, L. Budin, *Parallelization of Elimination Tournament Selection without Synchronization*, Proc. 5th Int. Conf. on Intelligent Engineering Systems INES 2001, Helsinki, September 16-18., pp. 85-90., 2001.

[Gol 96] M. Golub, *Vrednovanje uporabe genetskih algoritama za aproksimaciju vremenskih nizova*, magistarski rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 1996.

[Gol 98] M. Golub, D. Jakobović, *A Few Implementations Of Parallel Genetic Algorithm*, Proc. 20th Int. Conference ITI'98, Pula, June 14-17 1998, pp.332-337

[Gre 01] William A. Greene, *Dynamic Load-Balancing via a Genetic Algorithm,* 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01) November 07 - 09, 2001 Dallas, Texas, http://csdl.computer.org/comp/proceedings/ictai/2001/1417/00/14170121abs.htm

[Han 04] James V. Hansen, *Genetic search methods in air traffic control,* Computers and Operations Research, v 31, n 3, March, 2004, p 445-459, http://www.sciencedirect.com/

[Hay 96] T. D. Haynes, D. A. Schoenefeld, R. L. Wainwright, *Type Inheritance in Strongly Typed Genetic Programming*, Advances in Genetic Programming II, P. J. Angeline, K. E. Kinnear, (eds), MIT Press, 1996.

[He 03] Xiaoshan He, Xian-He Sun, Gregor Von Laszewski, *A QoS Guided Scheduling Model in Grid Environment*, Journal of Computer Science and Technology, Volume 18 , Issue 4 (July 2003), pp. 442 – 451 http://www.cs.iit.edu/~scs/psfiles/jcst_XHe-5-28.pdf

[Hel 02] Terry M. Helm, Steve W. Painter, Robert Oakes, *A comparison of three optimization methods for scheduling maintenance of high cost, long-lived capital assets,* Winter Simulation Conference Proceedings, v 2, 2002, p 1880-1884

[Hin 97] Robert Hinterding, Zbigniew Michalewicz, Agoston E. Eiben, *Adaptation in Evolutionary Computation: A Survey,* IEEECEP: Proceedings of The IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, 1997. http://citeseer.ist.psu.edu/hinterding97adaptation.html

[Iba 77] Oscar H. Ibarra, Chul E. Kim, *Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors*, Journal of the ACM, Volume 24 , Issue 2 (April 1977), pp. 280 – 289 http://portal.acm.org/citation.cfm?id=322011&jmp=abstract&dl=GUIDE&dl=ACM

[Jak 97] D. Jakobović, *Adaptive Genetic Operators in Elimination Genetic Algorithm*, Proc. 19th Int. Conference ITI'97, Pula, June 17-20 1997, pp.351-356

[Jak 98] D. Jakobović, M. Golub, *Adaptive Genetic Algorithm*, Proc. 20th Int. Conference ITI'98, Pula, June 14-17 1998, pp.351-356

[Jak 99] D. Jakobović, M. Golub, *Adaptive Genetic Algorithm*, Journal of Computing and Information Technology CIT, Vol. 7, No. 3, September 1999., pp. 229-236

[Jon 98] Albert Jones, Luis C. Rabelo, *Survey of Job Shop Scheduling Techniques,* NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, 1998., http://www.nist.gov/msidlibrary/summary/9820.html

[Kas 99] Joachim Käschel, Tobias Teich, Gunnar Köbernik, Bernd Meier, *Algorithms for the Job Shop Scheduling Problem - a comparison of different methods*, European Symposium on Intelligent Techniques ESIT ´99, June 3-4, 1999, Orthodox Academy of Crete, Greece http://www.erudit.de/erudit/events/esit99/12553_P.pdf

[Kin 93] Kenneth E. Kinnear, *Evolving a Sort: Lessons in Genetic Programming,* Proceedings of the 1993 International Conference on Neural Networks, Vol. 2, pp. 881-888, IEEE Press, 28 March -1 April 1993., ftp://cs.ucl.ac.uk/genetic/ftp.io.com/papers/kinnear.icnn93.ps.Z

[Koz 03] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*, Kluwer Academic Publishers, 2003.

[Koz 90] 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,

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

[Koz 90a] John R. Koza, *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems,* Stanford University Computer Science Department technical report STAN-CS-90-1314. June 1990.,
http://www.genetic-programming.com/jkpubs72to93.html

[Koz 92] J. R. Koza, *Genetic Programming – On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[Koz 94] J. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press, 1994.

[Koz 95] J. Koza, *Genetic Programming and Hill Climbing*, Machine Learning List, Vol. 7, No. 14, 18.9.1995.
http://www.ics.uci.edu/~mlearn/MLlist/v7/14.html

[Lan 00] W. B. Langdon, W. Banzhaf, *Genetic Programming Bloat without Semantics,* Parallel Problem Solving from Nature - PPSN VI 6th International Conference, LNCS, Vol. 1917, pp. 201-210, Springer Verlag, 16-20 September 2000.,
ftp://cs.ucl.ac.uk/genetic/papers/wbl_ppsn2000.ps.gz

[Lan 02] W. B. Langdon, R. Poli, *Foundations of Genetic Programming*, Springer-Verlag, 2002.

[Lan 05] William Langdon, Steven Gustafson, John Koza, *The Genetic Programming Bibliography*, 2005
http://liinwww.ira.uka.de/bibliography/Ai/genetic.programming.html

[Lan 97] W. B. Langdon, R. Poli, *Genetic Programming Bloat with Dynamic Fitness,* Technical Report, University of Birmingham, School of Computer Science, Number CSRP-97-29, 3 December 1997.,
ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1997/CSRP-97-29.ps.gz

[Lan 98] W. B. Langdon, *Genetic Programming and Data Structures*, Kluwer Academic Publishers, 1998.

[Lee 04] S. M. Lee, A.A. Asllani, *Job scheduling with dual criteria and sequence-dependent setups: mathematical versus genetic programming,* Omega, v 32, n 2, April 2004, p 145-53

[Lee 97] Young Hoon Lee, Kumar Bhaskaran, Michael Pinedo, *A heuristic to minimize the total weighted tardiness with sequence-dependent setups,* IIE Transactions, 29, 45-52, 1997.

[Lek 03] Lekin®, Flexible Job Shop Scheduling System
http://www.stern.nyu.edu/om/software/lekin/

[Leu 04] J. Y-T. Leung (ed.), *Handbook of scheduling,* Chapman & Hall/CRC, 2004.

[Leu 95] J. Y-T. Leung, *A survey of scheduling results for imprecise computation tasks*, Imprecise and approximate computation, Kluwer Academic Publishers, pp. 35-42, 1995

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

[Mar 04] Goran Martinović, *Postupci raspoređivanja u raznorodnim računalnim sustavima*, doktorska disertacija, Fakultet elektrotehnike i računarstva, Zagreb, 2004.

[Meg 05] Nicole Megow, Marc Uetz, Tjark Vredeveld, *Stochastic Online Scheduling on Parallel Machines,* G. Persiano and R. Solis-Oba (eds): Approximation and Online Algorithms, Lecture Notes in Computer Science 3351, pages 167-180, Springer, 2005.,
http://www.math.tu-berlin.de/~nmegow/muv05sos.pdf

[Mic 92] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutionary Programs*, Springer-Verlag, Berlin, 1992

[Miy 00] Kazuo Miyashita, *Job-Shop Scheduling with GP,* Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), pp. 505-512, Morgan Kaufmann, 10-12 July 2000.

[Moh 83] Ram Mohan, V. Rachamadugu, Thomas E. Morton, *Myopic Heuristics for the Weighted Tardiness Problem on Identical Parallel Machines,* Working Paper, The Robotics Institute, Carnegie-Mellon University, 1983.

[Mon 01] Patrick Monsieurs, Eddy Flerackers, *Detecting and Removing Inactive Code in Genetic Programs,* 2001,
http://alpha.luc.ac.be/~lucp1089/DetectingAndRemovingInactiveCode.pdf

[Mon 95] D. J. Montana, *Strongly Typed Genetic Programming*, Evolutionary Computation, 3(2):199-230, 1995.

[Mor 93] Thomas E. Morton, David W. Pentico, *Heuristic Scheduling Systems,* John Wiley & Sons, Inc., 1993.

[Nei 99] Michael O'Neill, Conor Ryan, *Automatic Generation of Caching Algorithms,* Evolutionary Algorithms in Engineering and Computer Science, pp. 127-134, John Wiley & Sons, 30 May - 3 June 1999., http://www.mit.jyu.fi/eurogen99/papers/oneill.ps

[Nei 99a] Michael O'Neill, Conor Ryan, *Automatic Generation of Programs with Grammatical Evolution*, 1999, http://citeseer.ist.psu.edu/276159.html

[Nor 94] P. Nordin, *A Compiling Genetic Programming System that Directly Manipulates the Machine Code*, Advances in Genetic Programming, pp. 311-331, MIT Press, 1994

[Nor 95] P. Nordin, W. Banzhaf, *Genetic Programming Controlling a Miniature Robot*, Working Notes for the AAAI Symposium on Genetic Programming, pp. 61-67, MIT, Cambridge, 1995.

[Nov 03] Sonja Novković, Davor Šverko, *A Genetic Algorithm With Self-Generated Random Parameters,* Journal of Computing and Information Technology - CIT, Vol. 11, No. 4, December 2003., pp. 271-284

[Ok 00] S. Ok, K. Miyashita, S. Nishihara, *Improving Performance of GP by Adaptive Terminal Selection,* Proc. of the Pacific Rim International Conference on Artificial Intelligence (PRICAI), pp.435-445, 2000, http://staff.aist.go.jp/k.miyashita/publications/PRICAI2000.ps

[Ok 01] S. Ok, K. Miyashita, K. Hase, *Evolving Bipedal Locomotion with Genetic Programming --- Preliminary Report,* Proc. of the Congress on Evolutionary Computation 2001, pp.1025-1032, 2001, http://staff.aist.go.jp/k.miyashita/publications/cec.ps

[Pat 97] Norman Paterson, Mike Livesey, *Evolving caching algorithms in C by genetic programming,* Genetic Programming 1997: Proceedings of the Second Annual Conference, pp. 262-267, Morgan Kaufmann, 13-16 July 1997., 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.sciencedirect.com/

[Pfu 08] Michele Pfund, John W. Fowler, Amit Gadkari, Yan Chen, *Scheduling jobs on parallel machines with setup times and ready times*, Computers & Industrial Engineering 54 (2008) 764–782

[Pin 04] M. Pinedo, *Offline Deterministic Scheduling, Stochastic Scheduling, and Online Deterministic Scheduling: A Comparative Overview*, Handbook of Scheduling, J. Y-T. Leung (ed.), Chapman & Hall/CRC, 2004.

[Pol 99] Riccardo Poli, *Parallel Distributed Genetic Programming,* New Ideas in Optimization, McGraw-Hill, 1999., http://citeseer.ist.psu.edu/328504.html

[Pru 04] K. Pruhs, J. Sgall, E. Torng, *Online scheduling*, Handbook of Scheduling, J. Y-T. Leung (ed.), Chapman & Hall/CRC, 2004.

[Rus 97] R. M. Russell, J. E. Holsenback, *Evaluation of greedy, myopic and less-greedy heuristics for the single machine, total tardiness problem,* Journal of the Operational Research Society (1997), 48, 640-646

[Sch 94] E. Schöneburg, F. Heinzmann, S. Feddersen, *Genetische Algorithmen und Evolutionsstrategien*, Addison-Wesley, 1994.

[Ser 01] F. Seredynski, J. Koronacki, C. Z. Janikow, *Distributed multiprocessor scheduling with decomposed optimization criterion,* Future Generation Computer Systems, Volume 17, Issue 4 , January 2001, Pages 387-396, http://www.sciencedirect.com/

[Ser 99] F. Seredynski, J. Koronacki, C. Z. Janikow, *Distributed Scheduling with Decomposed Optimization Criterion: Genetic Programming Approach,* Internationa Parallel and Distributed Processing Symposium, workshop: Bio-Inspired Solutions to Parallel Processing Problems, 1999., http://ipdps.eece.unm.edu/1999/biosp3/seredyns.pdf

[Sil 03] Sara Silva, Jonas Almeida, *Dynamic Maximum Tree Depth - A Simple Technique for Avoiding Bloat in Tree-Based GP ,* Proc. of the Genetic and Evolutionary Computation Conference (GECCO-2003), pp. 1776-1787, Genetic and Evolutionary Computation Conference (GECCO-2003), Chicago, Illinois USA, July-2003, http://cisuc.dei.uc.pt/ecos/view_pub.php?id_p=109

[Sou 98] T. Soule, *Code Growth in Genetic Programming,* PhD Thesis, University of Idaho, 1998., http://www.cs.uidaho.edu/~tsoule/research/the3.ps

[Sri 94] M. Srinivas, L. M. Patnaik, *Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms*, IEEE Trans. Systems, Man and Cybernetics, April 1994.

[Sri 94a] M. Srinivas, L. M. Patnaik, *Genetic Algorithms: A Survey*, IEEE Computer, June 1994.

[Tac 94] W. A. Tackett, *Recombination, Selection and the Genetic Construction of Computer Programs*, PhD thesis, University of Southern California, Department of Electrical Engineering Systems, 1994.

[Tai 03] E. Taillard, *Scheduling Instances*, 2003. 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, http://www.genetic-programming.org/sp2003/Talbott.pdf

[Tel 95] A. Teller, M. Veloso, *PADO: Learning Tree Structured Algorithms for Orchestration into an Object Recognition System*, Technical Report CMU-CS-95-101, Department of Computer Science, Carnegie Mellon University, 1995.

[Vaz 00] Manuel Vazquez, L. Darrell Whitley, *A Comparison of Genetic Algorithms for the Dynamic Job Shop Scheduling Problem,* Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '00), Las Vegas, Nevada, USA, July 8-12, 2000

[Wal 05] Scott S. Walker, Robert W. Brennan, Douglas H. Norrie, *Holonic Job Shop Scheduling Using a Multiagent System*, IEEE Intelligent Systems, 2/2005, pp. 50-57

[Wal 96] P. Walsh, C. Ryan, *Paragen: A Novel Technique for the Autoparallelisation of Sequential Programs Using Genetic Programming*, Genetic Programming 96: Proceedings of the 1st Annual Conference, pp. 406-409, MIT Press, 1996.

[Wan 03] J.-S. Wang, *Influences of Function Sets in Genetic Programming,* Genetic Algorithms and Genetic Programming at Stanford 2003, http://www.genetic-programming.org/sp2003/Wang.pdf

[Wol 97] D. H. Wolpert, W. G. Macready, *No Free Lunch Theorems for optimization*, IEEE Trans. on Evolutionary Computation, 1(1):67-82, 1997.

[Yin 03] Wen-Jun Yin, Min Liu, Cheng Wu, *Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming,* Proceedings of the 2003 Congress on Evolutionary Computation CEC2003, pp. 1050-1055, IEEE Press, 8-12 December 2003.,

[Zha 96] B.-T. Zhang, H. Mühlenbein, *Adaptive Fitness Functions for Dynamic Growing/Pruning, of Program Trees*, Advances in Genetic Programming 2, pogl. 12, pp.241-256, MIT Press, 1996.