# MAST90098
# Approximation Algorithms and Heuristics
# Project - 2016

Ria Szeredi, Kenneth Young, Lotte Romijn

October 9, 2016

**Makespan Scheduling Problem (MS)**

Schedule $n$ jobs with designated processing times on $m$ identical machines in such a way that the whole processing time is minimised.

**Input**:
Positive integers $p_1, p_2, ..., p_n$ and an integer $m \geq 2$ for some $n \in \mathbb{N} - \{0\}$. Each $p_i$ is the processing time of the $i$th job on any of the $m$ available machines.

**Constraints**:
For every input instance $(p_1, ..., p_n, m)$ of MS,
$\mathcal{M}(p_1, ..., p_n, m) = \{S_1, S_2, ..., S_m \mid S_i \subseteq \{1, 2, ..., n\}$ for $i = 1, ..., m$, $\cup_{k=1}^{m} S_k = \{1, 2, ..., n\}$, and
$S_i \cap S_j = \emptyset$ for $i \neq j\}$.

**Cost**:
For each $(S_1, ..., S_m) \in \mathcal{M}(p_1, ..., p_n, m)$,
$\text{cost}((S_1, ..., S_m), (p_1, ..., p_n, m)) = \max\{\sum_{l \in S_i} p_l | i = 1, ..., m\}$.

**Goal**:
*minimum*

# 1 Greedy Local Search Algorithm for MS

We implement a greedy local search (GLS) algorithm for MS, which picks the lowest cost neighbour at any iteration.

## GLS 1.a

We consider a *k-jump*-neighbourhood:

We define a mapping $f_X : \mathcal{M}(x) \to 2^{\mathcal{M}(x)}$, such that if $\beta = \{\beta_1, \beta_2, ..., \beta_m\} \in f_x(\alpha)$ for some $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_m\} \in \mathcal{M}(x)$ if $\beta$ can be obtained from $\alpha$ by $k$ jumps.

Each jump allows some job $i \in S_j$ for some $j \in \{1, ..., m\}$ to move to any one of the $m$ machines, including to the one in which it is contained. Hence, after one jump $\alpha_i \neq \beta_j$ for some $i \in \{1, ..., m\}$ and $j \in \{1, ..., m\}$, and $\alpha_k = \beta_k$ for $k \neq i$ and $k \neq j$, or $\alpha = \beta$ if the job jumps to its own machine.

We can check that the conditions for a suitable neighbourhood are satisfied. Let $\alpha = \{\alpha_1, ..., \alpha_m\}$ be an initial, feasible solution.

1. (*Reflexivity*): Since we allow a job to jump to the machine in which it is already contained, it is possible that, after $k$ jumps, all jobs are in their original machines, and the solution is identical to $\alpha$. Hence, we have that $\alpha \in f_X(\alpha)$ for every $\alpha \in \mathcal{M}(x)$. (We call this a 'loop').
2. (*Symmetry*): If $\beta \in f_X(\alpha)$ for some $\alpha \in \mathcal{M}(x)$, then $\alpha \in f_X(\beta)$, since reversing the $k$ jumps from which $\beta$ was obtained from $\alpha$ is also a $k$ jump.
3. (*Connectivity*): If we have any two feasible solutions $\alpha$ and $\beta$, we can always obtain one from the other by performing a finite number of $k$ jumps (since we allow a job to jump to any machine). If $k \geq n$, one $k$ jump suffices, since we allow loops. If $k \leq n$, we perform maximally $\lceil \frac{n}{k} \rceil$ $k$ jumps. Hence, for all $\alpha, \beta \in \mathcal{M}(x)$, there exists a positive integer $k$ and $\gamma_1, ..., \gamma_k \in \mathcal{M}(x)$ such that $\gamma_1 \in f_X(\alpha)$, $\gamma_{i+1} \in f_X(\gamma_i)$ for $i = 1, ..., k-1$, and $\beta \in f_X(\gamma_k)$.

## GLS 1.b

We implement the *greedy local search* (GLS) algorithm in Python. We will call the resulting total processing time, produced by the algorithm, the 'makespan value'. See attached code files .

# GLS 1.c

We perform an experimental study on randomly generated instances of MS across a range of $n$ and $m$ values in order to select an appropriate value for $k$.

We consider $n = 15, 20, 25, 30, 35, 40, 45, 50$, $m = 2, 4, 6, 8, 10$, and $k = 1, 2, 3$. For each combination of $n$, $m$ and $k$, we generate 1 instance of MS. For each instance, we randomly select the processing times $p_i$, for $i = 1, ..., n$, uniformly distributed between 1 and 100. We determine the runtime of our GLS implementation for each instance, and the solution quality by comparing the makespan value to a lower bound. We define this lower bound to be the least possible makespan value if we were able to process fractions of the jobs on any machine, i.e.

$$\text{lower bound} = \frac{\sum_{i=1}^{n} p_i}{m}. \tag{1}$$

We subsequently calculate the solution quality, denoting the makespan value produced by GLS as 'makespan',

$$\text{'makespan gap'} = \frac{\text{makespan}}{\text{lower bound} - 1} \times 100\%, \tag{2}$$

which represents the percentage that makespan value is above its lower bound
The results of 1 realization per $n$, $m$, and $k$ are given in Table 1 and Figure 1. In Figure 1, we show three-dimensional plots of $n$ versus $m$ versus the makespan gap (in percentage), in Figure 1a, 1c, and 1e, and $n$ versus $m$ versus the run time (in seconds), in Figure 1b, 1d, and 1f. For clarity, we only plot values for the makespan between 0 and 20 %, and values for the run time between 0 and 300 seconds. We consider a 'reasonable' runtime to be approximately 5 minutes (300 seconds).

We can see in Table 1a, 1c, and 1e that for all $n$ and $m$, the makespan gap decreases as we increase $k$. For $k = 1$, the average makespan gap for all $n$ and $m$ in Table 1a is 6.98 %. For $k = 2$, the average makespan gap for all $n$ and $m$ in Table 1c is 3.68 %. For $k = 3$, the average makespan gap for all $n$ and $m$ in Table 1e is 1.32 %. The makespan gap is larger for small $n$ and large $m$, i.e. when few jobs need to be distributed over many machines. It is especially larger when $n$ is close to $m$. SAY WHY

In Table 1b, 1d, and 1f, we see that for all $n$ and $m$, the run time increases as $k$ increases. It increases exponentially: we see that the average run time is 0.072 s for $k = 1$, 19.79 s for $k = 2$, and 1199 s for $k = 3$. The runtime is larger for large $m$ and large $n$, as in this case there are many possibilities for jobs to make a jump, and hence, we have a large neighbourhood.

For our remaining experiments, we choose $k = 2$. In Figure 1d, we see that for all instances we have considered so far, the run time is smaller than 5 minutes (300 seconds). Moreover, the makespan gap is

smaller than for $k = 1$, for all instances except for $n = 15$ and $m = 8, 10$, i.e. when the number of jobs is relatively close to the number of machines. Hence, we make a trade-off between solution quality and run time, and select $k = 2$ as our $k$-value.

# GLS 1.d

We perform an experimental study to test the performance of the GLS algorithm, using $k = 2$. We test for $n = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100$ and $m = 2, 4, 6, 8, 10$.

For each combination of $n$, $m$ and $k$, we generate 10 instances of MS. For each instance, we randomly select the processing times $p_i$, for $i = 1, ..., n$, uniformly distributed between 1 and 100. Each instance is generated with a different random seed.

We now also use different initial solutions; before starting the GLS algorithm, we need to find an initial, feasible solution for the instance. We do this by randomly putting the jobs in any machine ('random'), or by using the greedy makespan algorithm ('GMS'). The GMS works by first sorting the jobs by non-increasing processing time, assigning the $m$ largest jobs each to a different machine, and assigning the remaining jobs to the least loaded machine.

We consider a 'reasonable' runtime to be approximately 5 minutes. The averaged results of 10 realizations per $n$, $m$, and $k$ are presented in Table 2 and Figure 2. In Figure 2, only makespan gap values between 0 and 20 % are plotted, and run time values between 0 and 300 s (5 minutes).

From Table 2a, we can calculate that the average makespan gap over all $n$ and $m$ for the random initial solution is 4.09 %. In Table 2c, the average makespan gap over all $n$ and $m$ for the random initial solution is 4.11 %. In Table 2b, the average run time over all $n$ and $m$ for the random initial solution is 176.74 s. In Table 2d, the average run time over all $n$ and $m$ for the GMS initial solution is 171.04 s. Hence, the random initial solution and the GMS initial solution give similar results. WHY Later we will also use these two initial solutions when testing VDS and our heuristic.

## GLS experiments with random initial solution

### $k = 1$

(a) Makespan gap

| m \ n | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|
| 2 | 1.40 | 0.35 | 0.34 | 0.15 | 0.14 | 0.09 | 0.04 | 0.07 |
| 4 | 5.93 | 2.84 | 2.51 | 1.43 | 1.56 | 0.84 | 0.88 | 0.69 |
| 6 | 11.99 | 10.06 | 5.62 | 3.73 | 2.74 | 3.00 | 2.58 | 1.93 |
| 8 | 17.85 | 12.18 | 13.44 | 12.42 | 8.11 | 6.45 | 5.03 | 3.81 |
| 10 | 40.01 | 19.70 | 14.06 | 18.40 | 14.12 | 9.98 | 12.30 | 10.50 |

(b) Run time

| m \ n | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|
| 2 | 0.001 | 0.002 | 0.003 | 0.003 | 0.005 | 0.005 | 0.008 | 0.009 |
| 4 | 0.004 | 0.008 | 0.011 | 0.018 | 0.024 | 0.032 | 0.045 | 0.044 |
| 6 | 0.013 | 0.019 | 0.030 | 0.043 | 0.058 | 0.067 | 0.093 | 0.096 |
| 8 | 0.020 | 0.036 | 0.043 | 0.075 | 0.102 | 0.150 | 0.217 | 0.219 |
| 9 | 0.026 | 0.060 | 0.081 | 0.135 | 0.179 | 0.200 | 0.325 | 0.385 |

### $k = 2$

(c) Makespan gap

| m \ n | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|
| 2 | 0.14 | 0 | 0.08 | 0.06 | 0 | 0.05 | 0.04 | 0.04 |
| 4 | 0.12 | 0.22 | 0.21 | 0.06 | 0.17 | 0.09 | 0.04 | 0 |
| 6 | 4.87 | 1.11 | 2.59 | 1.49 | 0.53 | 0.37 | 0.24 | 0.30 |
| 8 | 32.00 | 7.36 | 0.85 | 0.95 | 0.78 | 0.76 | 0.42 | 0.76 |
| 10 | 65.48 | 7.61 | 8.24 | 3.38 | 2.89 | 1.53 | 0.56 | 0.83 |

(d) Run time

| m \ n | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|
| 2 | 0.02 | 0.05 | 0.05 | 0.07 | 0.11 | 0.45 | 0.72 | 1.09 |
| 4 | 0.18 | 0.78 | 0.64 | 0.73 | 2.40 | 4.68 | 7.40 | 5.21 |
| 6 | 0.50 | 0.87 | 3.27 | 2.82 | 8.53 | 19.86 | 11.44 | 29.11 |
| 8 | 1.02 | 3.38 | 8.65 | 17.52 | 21.71 | 42.08 | 49.94 | 94.56 |
| 10 | 2.30 | 5.59 | 10.94 | 23.56 | 59.58 | 53.83 | 100.87 | 195.12 |

### $k = 3$

(e) Makespan gap

| m \ n | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|
| 2 | 0.14 | 0 | 0 | 0 | 0 | 0 | 0 | 0.04 |
| 4 | 0.20 | 0.41 | 0.14 | 0.12 | 0 | 0.08 | 0.08 | 0.11 |
| 6 | 2.45 | 1.65 | 0.56 | 0.12 | 0.10 | 0 | 0.08 | 0.04 |
| 8 | 12.88 | 4.13 | 0.42 | 0.37 | 0.96 | 0.42 | 0.40 | 0.41 |
| 10 | 17.59 | 2.27 | 3.09 | 1.10 | 0.86 | 0.93 | 0.40 | 0.26 |

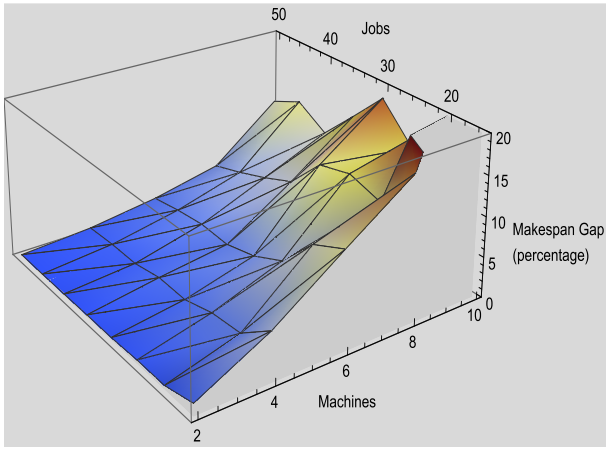
(f) Run time

| m \ n | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
|---|---|---|---|---|---|---|---|---|
| 2 | 0.09 | 0.24 | 1.37 | 1.62 | 3.98 | 11.04 | 16.93 | 25.06 |
| 4 | 1.21 | 7.75 | 14.09 | 34.79 | 42.06 | 121.77 | 291.35 | 505.66 |
| 6 | 7.90 | 25.30 | 64.16 | 131.15 | 244.04 | 871.73 | 1262.60 | 2590.84 |
| 8 | 19.78 | 80.02 | 194.86 | 435.55 | 987.58 | 2624.76 | 3264.98 | 5114.31 |
| 10 | 54.73 | 153.17 | 347.68 | 604.12 | 2002.52 | 4088.55 | 6251.35 | 15462.72 |

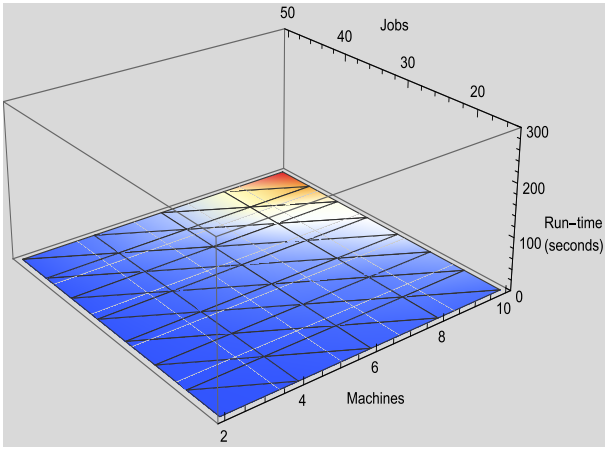
Table 1: GLS experiments: $k = 1, 2, 3$, using a random initial solution. The makespan gap is given in % and the run time in seconds.

# GLS experiments

$k = 1$

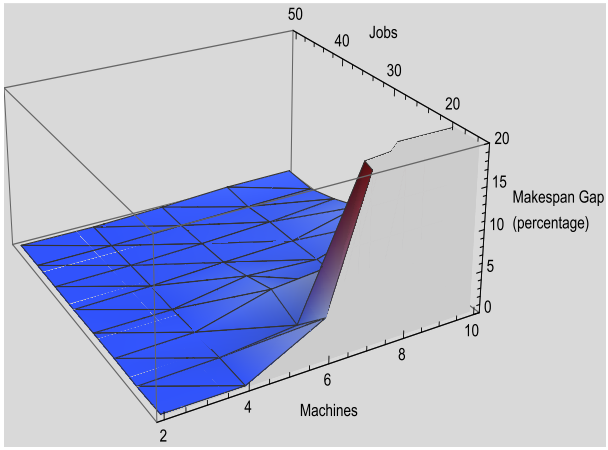



(a)

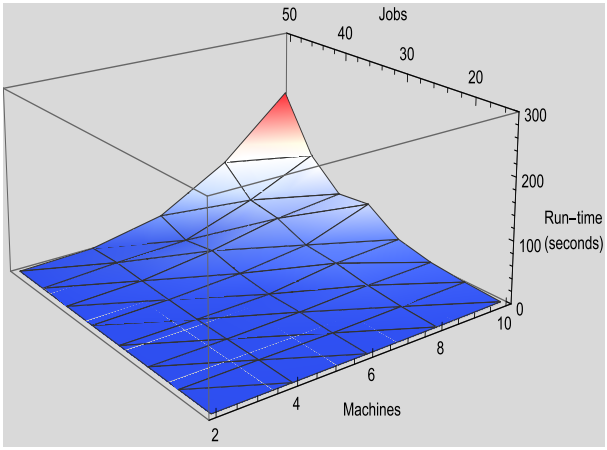



(b)

$k = 2$

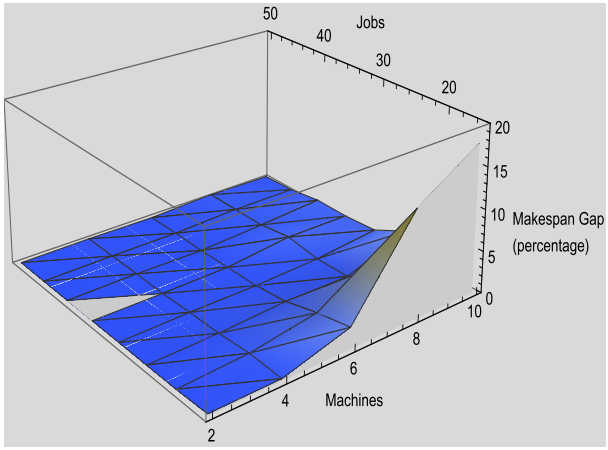



(c)

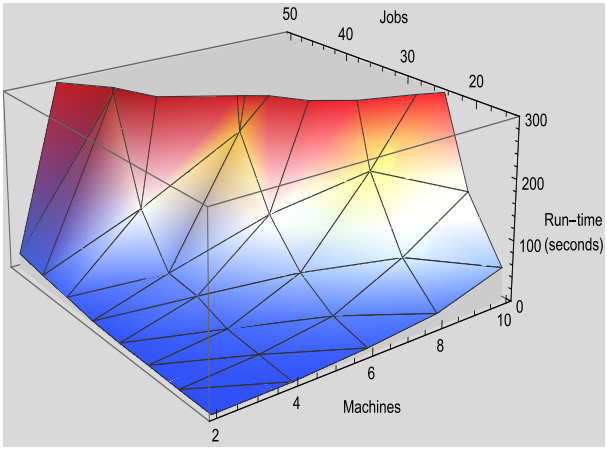



(d)

$k = 3$




(e)




(f)

Figure 1: **GLS Experiments:** $k = 1, 2, 3$**, using a random initial solution.**

# GLS experiments

## Random initial solution

(a) Makespan gap

| m \ n | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.56 | 0.06 | 0.05 | 0 | 0.02 | 0.02 | 0.02 | 0.01 | 0.02 | 0.01 |
| 4 | 6.23 | 0.54 | 0.22 | 0.10 | 0.10 | 0.05 | 0.02 | 0.04 | 0.02 | 0.02 |
| 6 | 17.52 | 1.05 | 0.36 | 0.29 | 0.15 | 0.15 | 0.07 | 0.09 | 0.10 | 0.07 |
| 8 | 66.76 | 5.64 | 1.12 | 0.50 | 0.27 | 0.37 | 0.17 | 0.12 | 0.14 | 0.15 |
| 10 | 86.32 | 9.86 | 2.13 | 0.87 | 0.64 | 0.33 | 0.26 | 0.32 | 0.27 | 0.20 |

(b) Run time

| m \ n | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.00 | 0.02 | 0.11 | 0.19 | 0.54 | 1.78 | 4.51 | 5.66 | 7.93 | 14.52 |
| 4 | 0.04 | 0.23 | 0.84 | 2.22 | 4.81 | 8.02 | 19.77 | 62.60 | 82.57 | 131.87 |
| 6 | 0.78 | 0.80 | 3.09 | 9.10 | 25.45 | 51.87 | 116.10 | 206.22 | 322.23 | 456.87 |
| 8 | 10.07 | 5.73 | 13.29 | 21.49 | 61.11 | 97.83 | 198.89 | 427.29 | 640.75 | 1117.00 |
| 10 | 32.48 | 23.80 | 39.58 | 98.76 | 185.58 | 360.85 | 659.68 | 533.94 | 1154.13 | 1614.11 |

## GMS initial solution

(c) Makespan gap

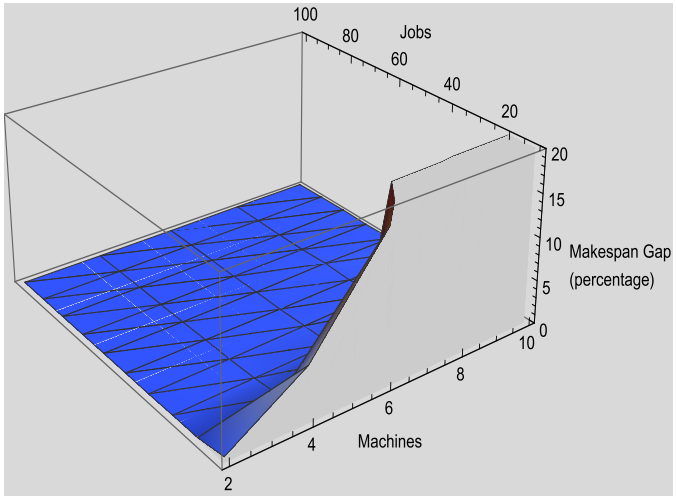
| m \ n | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.52 | 0.06 | 0.05 | 0 | 0.02 | 0.02 | 0.02 | 0.01 | 0.02 | 0.01 |
| 4 | 6.44 | 0.35 | 0.16 | 0.08 | 0.07 | 0.03 | 0.02 | 0.04 | 0.02 | 0.02 |
| 6 | 17.52 | 1.54 | 0.57 | 0.29 | 0.17 | 0.07 | 0.10 | 0.10 | 0.07 | 0.07 |
| 8 | 66.76 | 6.27 | 0.85 | 0.57 | 0.36 | 0.27 | 0.14 | 0.12 | 0.09 | 0.09 |
| 10 | 86.32 | 9.75 | 2.33 | 1.07 | 0.45 | 0.36 | 0.26 | 0.34 | 0.21 | 0.17 |

(d) Run time

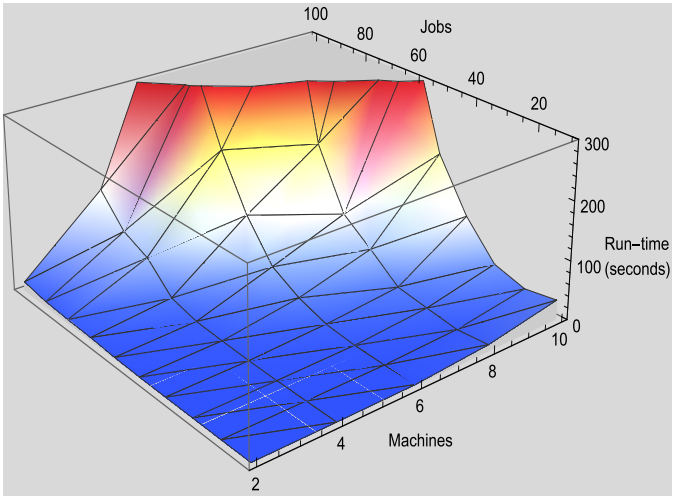
| m \ n | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 0.00 | 0.02 | 0.11 | 0.17 | 1.17 | 0.92 | 3.56 | 6.89 | 14.87 | 11.88 |
| 4 | 0.03 | 0.16 | 0.56 | 1.9 | 3.43 | 7.78 | 10.23 | 58.41 | 60.2 | 153.82 |
| 6 | 0.88 | 1.12 | 3.21 | 6.83 | 20.65 | 47 | 99.3 | 166.63 | 203.52 | 209.51 |
| 8 | 5.42 | 8.8 | 23.13 | 15.65 | 38.21 | 62.5 | 175.91 | 530.53 | 602.13 | 1157.53 |
| 10 | 23.73 | 35.67 | 37.07 | 84.23 | 167.81 | 399.51 | 345.57 | 697.98 | 1457.24 | 1588.73 |

Table 2: Experiments GLS: k=2, using random and GMS initial solution. The makespan gap is given in % and the run time in seconds.

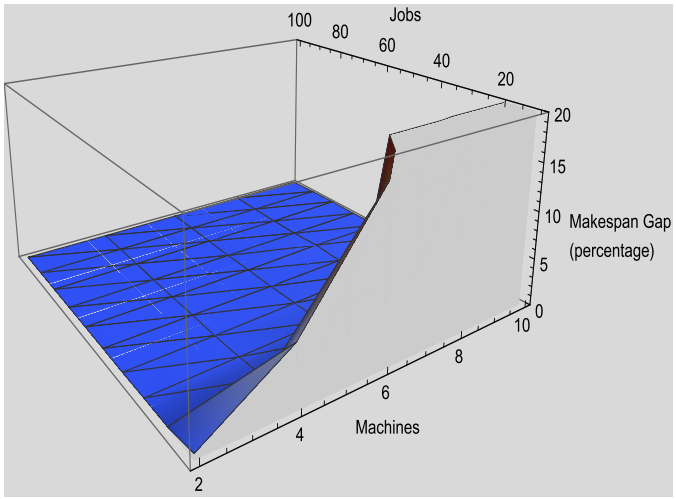
# GLS experiments



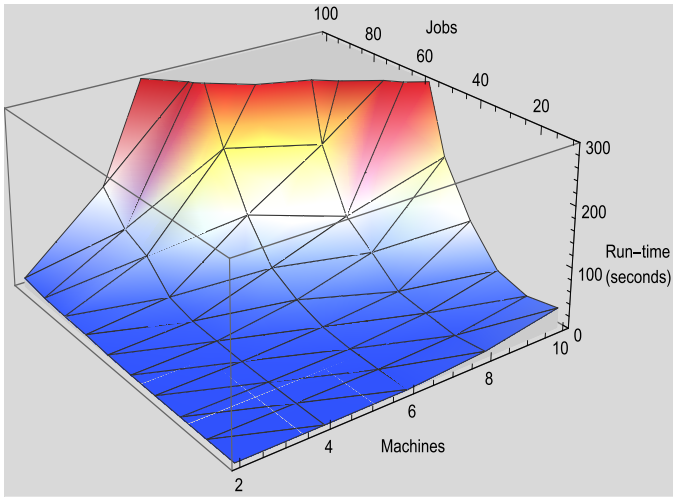

(a) Random




(b) Random




(c) GMS




(d) GMS

Figure 2: **Experiments GLS: k=2, using random and GMS initial solution.** (a) and (b) display the makespan gap (in percentage) and the run time (in seconds), using a random initial solution. (c) and (d) show the makespan gap (in percentage) and the run time (in seconds), using the GMS initial solution. We only plot makespan gap values between 0 and 20 %, and run time values between 0 and 300 s.

# 2 Variable Depth Search Algorithm for MS

In our Python code, we generally represent any feasible solution $\{S_1, S_2, ..., S_m\}$ to the Makespan Scheduling problem as a list $\xi = [\xi_1, \xi_2, ..., \xi_n]$ of length $n$, where each $\xi_i \in [1, ..., m]$ which represents in which machine job $i$ is contained. Methods have been included which convert the solutions between these two representations (*convertSol_toListOfMachines* and *convertSol_toSetsOfJobs*).

Pseudo-code for VDS: We begin with an input instance $I = (p_1, p_2, ..., p_n, m)$ of the Knapsack problem.

1. We generate an initial feasible solution $\xi = \{\xi_1, \xi_2, ..., \xi_n\}$, MENTION HOW
2. IMPROVEMENT := TRUE
3. EXCHANGE := $\{1, 2, ..., n\}$, $J = 0$, $\alpha_J := \xi$
4. **while** IMPROVEMENT = TRUE **do begin**
   (a) **while** EXCHANGE $\neq \emptyset$ **do**
      i. **begin** $J := J + 1$
      ii. $\alpha_J :=$ the best neighbour in the $k$-neighbourhood of $\alpha_{J-1}$. We define $gain(\alpha_{J-1}, \delta) =$ makespan$(\alpha_{J-1})$ − makespan$(\delta)$. Hence, the best neighbour $\alpha_J$ is defined such that $gain(\alpha_{J-1}, \alpha_J)$ =max$\{gain(\alpha_{J-1}, \delta) | \delta \in \text{Neigh}_k(\alpha_{J-1}) - \{\alpha_{J-1}\}$ and $\delta$ differs from $\alpha_{J-1}$ in the parameters of EXCHANGE only $\}$.
      iii. EXCHANGE := EXCHANGE − {the parameters in which $\alpha_J$ and $\alpha_{J-1}$ differ } (i.e., we remove the job index for which the machine has been changed from EXCHANGE).
   (b) **end**
   (c) Compute $gain(\alpha, \alpha_i) =$ makespan$(\alpha)$ − makespan$(\alpha_i)$, for $i = 1, ..., J$
   (d) Compute $l \in \{1, ..., J\}$ such that
   $gain(\alpha, \alpha_l) = \max\{gain(\alpha, \alpha_i) \, | i \in \{1, 2, ..., J\}\}$
   (e) **if** $gain(\alpha, \alpha_l) > 0$ **then**
      i. EXCHANGE := $\{1, 2, ..., n\}$
      ii. **end**
   (f) **else** IMPROVEMENT := FALSE
   (g) **end**
5. **output**$(\alpha)$

Say something about 'differentsolrequired'

# 3 Simulated Annealing Algorithm for MS

The heuristic we chose to implement is simulated annealing. In this section, we present pseudocode for the algorithm, as well as details about the generation of the initial value of the temperature. We also present experimental results for the same test instances used in parts 1(d) and 2(c). Finally, we conclude with a comparison to the performance of the simulated annealing heuristic to the performance of the GLS and VDS algorithms.

## Simulated Annealing Algorithm Pseudocode

In our algorithm, we make a slight modification to keep track of the best solution found by the algorithm. This is compared to the final solution found by the simulated annealing algorithm. We do this in order to avoid the algorithm returning a worse solution than one previously found.

We again use the GMS algorithm (see section **??**) to generate an initial feasible solution.

*Input*: An input instance $I = (p_1, p_2, ..., p_n, m)$ of the Makespan Scheduling problem.

1. Generate an initial feasible solution $\xi = \{\xi_1, \xi_2, ..., \xi_n\}$, using GMS.
   $T = \text{getInitialTemp}(S, \chi_0, ...)$
2. $I := 0$
   $best := \xi$
3. **while** $T > 0$ **do**
   (a) **begin** Randomly select a neighbour $\beta \in Neigh_x(\alpha)$
   (b) **if** makespan($\beta$) $\leq$ makespan($\alpha$) **then**
      i. $\alpha := \beta$
   **else**
      i. Generate a random number $r$ uniformly in the range $(0, 1)$
      ii. **if** $r < e^{-\frac{\text{makespan}(\alpha) - \text{makespan}(\beta)}{T}}$ **then**
         $\alpha := \beta$
   (c) **if** makespan($\alpha$) $<$ makespan($best$) **then**
      $best := \alpha$
   (d) $I := I + 1$
   (e) $T := \text{reduceTemperature}(T, I)$
   **end while**

4. **if** makespan($\alpha$) < makespan($best$)
   **output**($\alpha$)
   **else**
   **output**($best$)

## Choice of the Initial Temperature

We use an algorithm by **??**, which generates an initial temperature based on ... . The pseudocode is as follows.

## Choice of the Temperature Reduction Function