Information about the java program "Jobscheduling to minimize the expected flowtime"

Written by Peter Kampstra and Jolanda Veldhuis.

May 10, 2004

1 Introduction

In the article "Job scheduling to minimize expected flowtime" R. Righter discusses an algorithm to find the optimal policy to schedule n jobs to m processors. For the course Caput OBP an implementation is made. The implementation is available at: http://www.cs.vu.nl/˜jfveldhu/jobvusion
Before we explain the usage of the program we will give a summary about the algorithm from the article of R. Righter. For the complete algorithm we refer to the article "Job scheduling to minimize expected flowtime" written by R. Righter and published in the journal "System & Control letters" volume 10 issue 4.

2 Algorithm

The algorithm presented in the article schedules n jobs on m processors non-preemptively to minimize the total expected holding cost (weighted flowtime). The processors are uniform, but non identical. Processor j has an exponentially distributed processing time with rate μ_j. Job i has c_i holding cost per time unit.
Assume that the processors are ordered by their speeds, $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_m$, and assume that the jobs are ordered by their holding costs, $c_1 \geq c_2 \geq \ldots \geq c_n$. Define policy π as the policy that takes the fastest (lowest indexed) available processor, say j and assigns job k to it if and only if $k > f(j) \geq k - 1$, where

$$f(j) = \frac{\sum_{i=1}^{j} \mu_i}{\mu_j} - j$$

After each assignment renumber the jobs (so that job $k+1$ becomes job k, etc.), and repeat the procedure with the next fastest available processor.

Define

$$J_k = max\{j : k > f(j)\},\ J_0 = 0$$

Since $\mu_{j-1} \geq \mu_j$, $f(j)$ is nondecreasing in j. This means also that $J_1 \leq J_2 \leq \ldots \leq J_k$. Thus, under policy π, job k will be processed by one of the processors in S_k where

$$S_k = \{J_{k-1} + 1, J_{k-1} + 2, \ldots, J_k\}$$

It is possible that S_k is empty ($J_k = J_{k-1}$). In this case job k will not be processed under policy π until one of the jobs $1, ..., k - 1$ is processed and job k becomes job $k - 1$.

As an example, suppose we have three processors with rates $\mu_1 = 3$, $\mu_2 = 2$, and $\mu_3 = 1$. Then $f(1) = 0$, $f(2) = 0.5$, and $f(3) = 3$ and $J_1 = J_2 = J_3 = 2$, $J_4 = \ldots = J_n = 3$, $S_1 = \{1, 2\}$, $S_4 = \{3\}$, and S_k is empty for $k \neq 1, 4$. In other words, the first job will be assigned to the fastest available of processors 1 and 2, job 2 and 3 will have to wait until job 1 is processed. The fourth job (if it exists) will be assigned to processor 3 if processor 3 is the fastest available processor. If there are less than 4 jobs then processor 3 will remain idle.

3 Implementation

We implemented the algorithm as discussed in section 2. We calculate the optimal schedule in four steps:

1. Sorting
After reading the input, we sort the variables μ and c. The available machines are ordened descending to μ and numbered. So, if you insert the following production rates: 2, 1, 3, then we assign machine 1 to the production rate 3, machine 2 to production rate 2 and machine 3 to production rate 1, so $\mu_1 = 3$, $\mu_2 = 2$, $\mu_3 = 1$. The same is done for the jobs. If you insert the following holding cost for the jobs: 5, 4, 6, then we assign the holding cost of 6 per time unit to job 1, 5 to job 2 and 4 to job 3, so $c_1 = 6$, $c_2 = 5$ and $c_3 = 4$.
We put the values in an array of doubles.

2. Calculate $f(j)$ for each machine
After sorting the input variables, we calculate the values for $f(j)$ with the use of the formula:

$$f(j) = \frac{\sum_{i=1}^{j} \mu_i}{\mu_j} - j$$

for all machines ($j = 1, ..., n$).

3. Calculate J_k for each job
As we already saw in section 2, the value J_k is defined as:

$$J_k = max\{j : k > f(j)\},\ J_0 = 0$$

The number J_k ranges from 0 to n, and can be easily calculated. Now each job k will be scheduled on the following set of machines: $S_k = \{J_{k-1} + 1, J_{k-1} + 2, ..., J_k\}$. If the set is empty (that is when $J_k = J_{k-1}$) then the job is not scheduled.

4. Calculate the expected holding time for each job
Let v_k be the total expected holding time of job k. The total expected holding time for a job can be calculated with the formula:

$$v_k = \frac{k + J_k}{\sum_{j=1}^{J_k} \mu_j}$$

The total expected holding cost if all processors are initially busy can be calculated with $V_\pi(\mathbf{c}) = \sum_{i=1}^{n} c_i v_i$ under the optimal policy π.

After these four steps the program prints the results on screen. The user is able to change the variables or clear them if he likes. The variables could easily be changed by a simple click in the textarea. To clear all the textareas just press the "Clear" button.

4 How to use the program?

There are two versions of the program. There is a java-applet version, this version is available on the website, and a command-line version. The source code and the compiled code is also available on the website mentioned at the beginning of the paper.
First we will discuss how to use the java-applet, after that we will discuss the command-line version.

4.1 Java-applet

If the user uses the java-applet to calculate the schedule, then the user first has to give some input values for the production rates (μ_i) and the holding costs (c_i). Figure 1 shows the startscreen of the program which already contains some inputvalues.

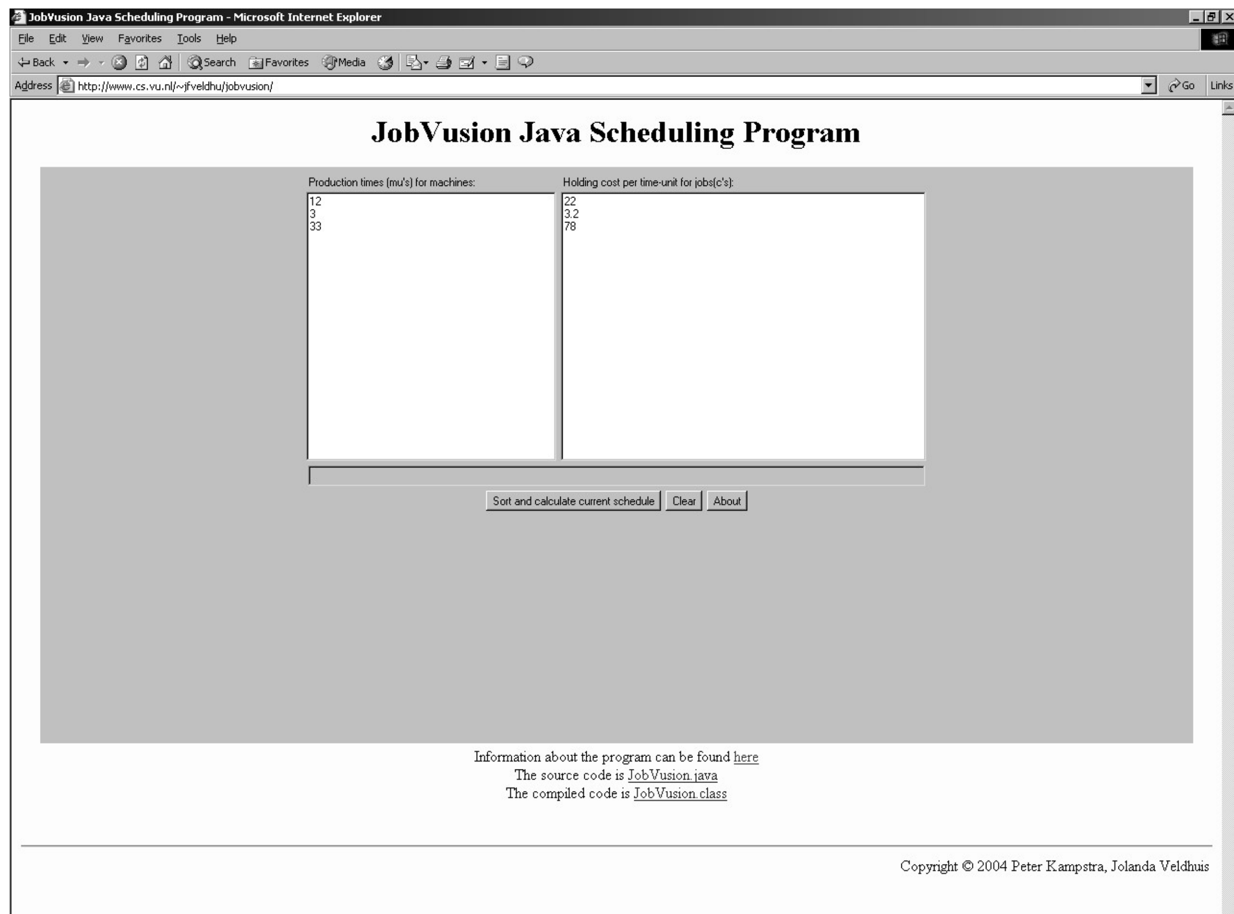

Figure 1: Startscreen of the java-applet.

In the textarea for the production rates (the textarea on the left), the user should enter the values for the production rates (μ_i) of the several machines. Each value should be separated by an enter, so there will be only one production

rate per line.
In the textarea for the holding costs (the textarea on the right), the user should enter the holding cost c_i for job i. Each line should contain one value.
When all values are inserted, the user should press the "Sort and calculate current schedule" button. Now the new schedule will be calculated. For our example the resultscreen will look like figure 2.

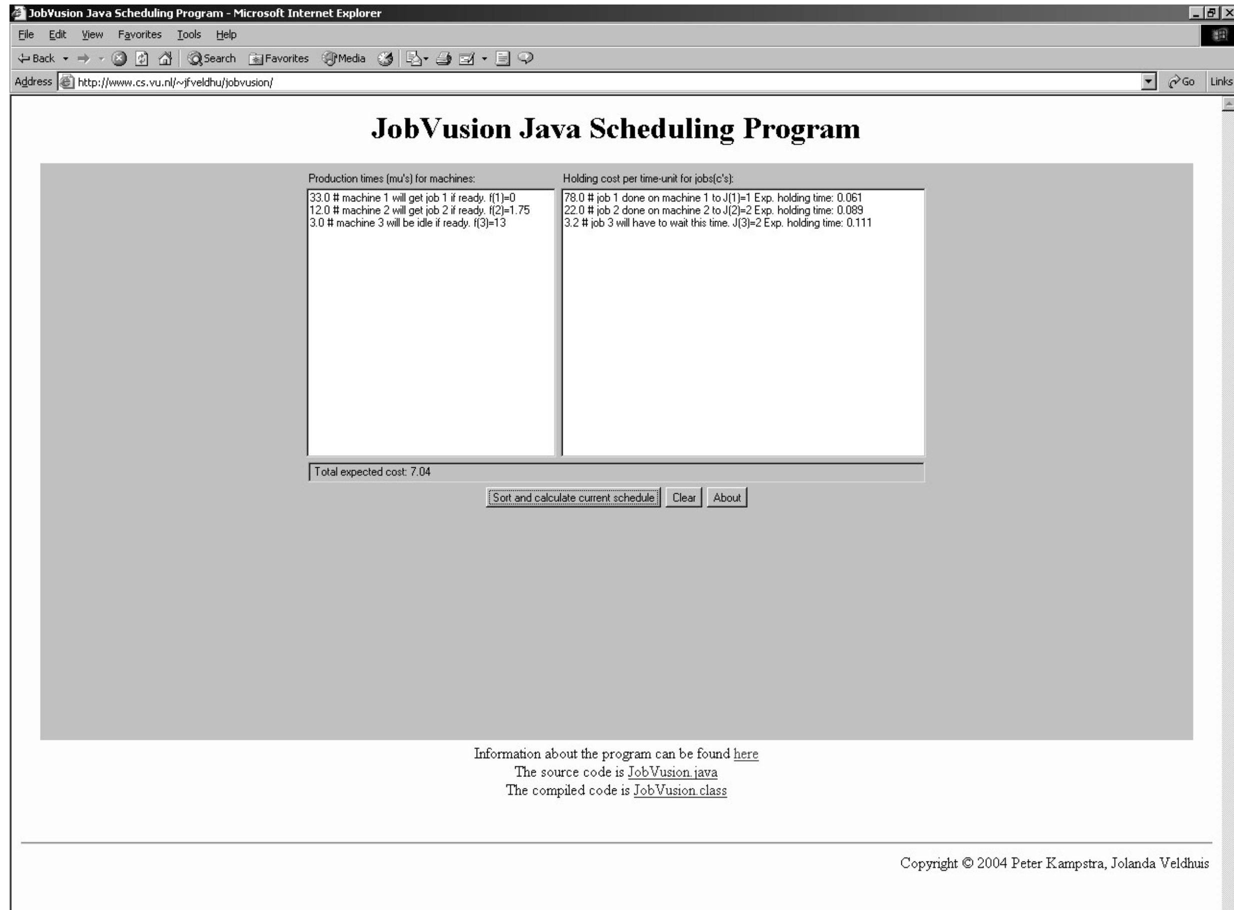

Figure 2: A possible result.

The program gives some machine results, job results and the total expected cost. The machine results are printed in the same textarea as were the user gave the inputvalues for the production rates. If we calculate the schedule for the defaultvalues as shown in figure 1, the machine results are equal to:

```
33.0 # machine 1 will get job 1 if ready.  f(1)=0
12.0 # machine 2 will get job 2 if ready.  f(2)=1.75
3.0 # machine 3 will be idle if ready.  f(3)=13
```

Notice that the machines are allocated to the production rates which are in descending order. This can be seen in the first part of an output line. For example, `33.0 # machine 1` means that production rate 33.0 is allocated to machine 1. The second part of an output line describes wich job should be produced first on that machine. So in our example machine 1 should produce job 1 first. It

is also possible that a machine stays idle. In our example machine 3 stays idle. The last part of an output line contains the value of $f(j)$.

The job results are printed in the same textarea as were the user gave the inputvalues for the holding costs. If we calculate the schedule for the default-values as shown in figure 1, the job results are equal to:

```
78.0 # job 1 done on machine 1 to J(1)=1 Exp.holding time:  0.061
22.0 # job 2 done on machine 2 to J(2)=2 Exp.holding time:  0.089
3.2 # job 3 will have to wait this time.J(3)=2 Exp.holding time:0.111
```

Notice that the jobs are allocated the same way as the machines. They are allocated to the holding costs which are in descending order. This can be seen in the first part of an output line. For example, `78.0 # job 1` means that holding cost 78.0 are allocated to job 1, so job 1 has holding cost 78 per time unit. The second part of the output line describes which job is done on which machines or that a job has to wait. For example, `job 1 done on machine 1 to J(1)=1` means that job 1 is done on machine 1.
`job 3 will have to wait this time` means that job 3 will not be processed until job 2 is processed, this because $J(2) = J(3) = 2$.
The expected holding time for each job is also calculated and printed per job. In the textarea under the textarea for the production rates and the holding costs the total expected cost are printed. In this example the total expected cost are equal to 7.04.

To clear all the fields and start all over again just press the "Clear" button.

The java-applet is written with compatability for JDK 1.1 (for Internet Explorer) and designed for easy cut-and-paste. It is able to schedule thousands of machines/jobs in a second.

4.2 Command-line

When starting the program in command-line the user is asked to give the total number of processors (machines) in the system and the corresponding production rates of this machines. Notice that each value should be separated by an enter.
For example, if we have 3 processors with the following production rates $\mu_1 = 3$, $\mu_2 = 2$, and $\mu_3 = 1$, the input is:

```
3
3
2
1
```

It isn't necessary to enter the production rates in descending order, the program will take care of that.
When the user has entered the values for the production rates the program asks for the total number of jobs and the corresponding holding costs. Again, each value should be separated by an enter.
For example, a possible input is:

```
4
5
2
```

```
9
3
```

So there are 4 jobs with the holding costs 5, 2, 9 and 3.
After giving the input the program will output the schedule which minimizes the expected flowtime. First the program outputs the machine results. In the case of our example the machine results are equal to:

```
3.0 # machine 1 will get job 1 if ready.  f(1)=0
2.0 # machine 2 will get job 1 if ready.  f(2)=0.5
1.0 # machine 3 will get job 4 if ready.  f(3)=3
```

Notice that the machines are allocated to the production rates which are in descending order. This can be seen in the first part of an output line. For example, `3.0 # machine 1` means that production rate 3.0 is allocated to machine 1. The second part of an output line describes wich job should be produced first on the machine. So in our example machine 1 should produce job 1 first. It is also possible that a machine stays idle. This should be the case if we only entered 3 jobs in our example instead of 4 jobs. The last part of an output line contains the value of $f(j)$. This value is interesting if the user would like to validate the results.

After giving the machine results, the program outputs the job results. In our example the job results are:

```
9.0 # job 1 done on machine 1 to J(1)=2
Expected holding time:  0.6
5.0 # job 2 will have to wait this time.  J(2)=2
Expected holding time:  0.8
3.0 # job 3 will have to wait this time.  J(3)=2
Expected holding time:  1
2.0 # job 4 done on machine 3 to J(4)=3
Expected holding time:  1.167
```

Notice that the jobs are allocated the same way as the machines. They are allocated to the holding costs which are in descending order. This can be seen in the first part of an output line. For example, `9.0 # job 1` means that holding cost 9.0 is allocated to job 1, so job 1 has holding cost 9 per time unit. The second part of the output line describes which job is done on which machines or that a job has to wait. For example, `job 1 done on machine 1 to J(1)=2` means that job 1 is done on machine 1 or on machine 2, that depends on which machine is available. `job 2 will have to wait this time` this means that job 2 will not be processed until job 1 is processed. In our example job 3 should also wait until job 1 and job 2 are processed.
The expected holding time for each job is also calculated and printed per job. The program ends its output with the total expected costs. In our example this is equal to 14.73.

5 Validation

We used several methods to validate the implementation of the algorithm. First we used the debug messages to be sure that the program reads all the input values correctly and that they were stored in the right way, so that the values were stored descending when necessary. After that we calculated all values manually to check whether the program works well.
Further we tested if the output of the program was logical and we tested some boundary problems. Some results of the testcases can be found below.

5.1 Case 1

Let's examine the case were there are 2 machines and 2 jobs in the system. Assume that $\mu_1 = 100$, $\mu_2 = 1$ and the holding costs for the jobs are the same. The program should assign all jobs to machine 1, because this machine is much faster than machine 2. Machine 2 should stay idle if ready. The program suffice this condition.

5.2 Case 2

Now we will examine the case were there are 1 machine and 2 jobs in the system. Let's assume that $c_1 = 200$, $c_2 = 1$. The expectation is that job 1 has priority over job 2, because it is much more expensive to hold job 1 than job 2. So job 1 should be produced first. The output of the system is as expected. The machine is assigned to produce job 1. Job 2 has to wait until the production of job 1 is finished.

5.3 Case 3

Now we consider a system with 4 machines and 3 jobs, with the values $\mu_1 = 3.000001$, $\mu_2 = 3.0$, $\mu_3 = 2.0$, $\mu_4 = 1.999999$ and $c_1 = 1.0$, $c_2 = 0.999999$, $c_3 = 0.000001$. The output should say that machine 1 and 2 produce job 1 if ready and machine 3 and 4 should produce job 2 if ready. The output should also state that job 1 will be produced on machine 1 to 2, job 2 on machine 3 to 4 and that job 3 has to wait. We examined this case because of the possibility that round off errors could occur. If that happens then the program handles different production times as the same, which is not allowed. The output of the program is just like we expected. There occur no round off problems while running the program.