RAIRO-Theor. Inf. Appl. 46 (2012) 329–342
DOI: 10.1051/ita/2011132

Available online at:
www.rairo-ita.org

JOB SHOP SCHEDULING WITH UNIT LENGTH TASKS

Meike Akveld[1] and Raphael Bernhard[2]

Abstract. In this paper, we consider a class of scheduling problems that are among the fundamental optimization problems in operations research. More specifically, we deal with a particular version called *job shop scheduling with unit length tasks*. Using the results of Hromkovič, Mömke, Steinhöfel, and Widmayer presented in their work *Job Shop Scheduling with Unit Length Tasks: Bounds and Algorithms*, we analyze the problem setting for 2 jobs with an unequal number of tasks. We contribute a deterministic algorithm which achieves a vanishing delay in certain cases and a randomized algorithm with a competitive ratio tending to 1. Furthermore, we investigate the problem with 3 jobs and we construct a randomized online algorithm which also has a competitive ratio tending to 1.

Mathematics Subject Classification. 68Q25.

1. Introduction

Many real-world applications ask for algorithms that cannot read the whole input before they may calculate the corresponding output. In these scenarios, called *online scenarios*, parts of the input arrive successively in certain time steps and after every such step, a so called *online algorithm* has to produce a piece of the output which cannot be altered afterwards. In the following, we give a more formal definition.

Definition 1.1. Let $x = (x_1, \ldots, x_n)$ be an input sequence. An *online algorithm* A computes $A(x) = (y_1, \ldots, y_n)$ where $y_i = A(x_1, \ldots, x_i)$ for $i \in \{1, \ldots, n\}$ which means that, for every $j > i$, y_i does not depend on x_j.

Keywords and phrases. Online algorithms, competitive analysis, job shop scheduling.

[1] Department of Mathematics, ETH Zürich, Zürich, Switzerland. akveld@math.ethz.ch

[2] Department of Information Technology and Electrical Engineering, ETH Zürich, Switzerland. beraphae@student.ethz.ch

Article published by EDP Sciences

In such setups, randomization is often used as a powerful tool which means that every chunk y_i of the output is calculated depending on $x_1, \ldots, x_i$ and some random string r. For a more detailed introduction of the concepts of online problems, we refer to the standard literature [5–7].

Let cost denote a *cost function* which maps every computed solution of an online algorithm to a real number. In this paper, we consider minimization problems, *i.e.*, we are interested in online algorithms A for which $\text{cost}(A(x))$ is as small as possible for all x. We analyze online algorithms using the concept of the *competitive ratio* which is the ratio between the cost of the computed solution and an optimal solution.

Definition 1.2. Let $c \geq 1$. An online algorithm A is called (strictly) *c-competitive*, if, for every input instance x, it holds that $\text{cost}(A(x)) \leq c \cdot \text{cost}(\text{Opt}(x))$ where Opt is an optimal solution for the instance x.

We look at a special class of randomized algorithms that randomly choose one algorithm from a finite set of deterministic algorithms (strategies). To measure the performance of such a randomized algorithm R, we are interested in the *expected competitive ratio*, *i.e.*, the sum of all possible competitive ratios achieved by the deterministic strategies multiplied by the probability that they are chosen by R.

Definition 1.3. Again, let $c \geq 1$ and x and Opt as above. A randomized online algorithm R is called strictly $E[c]$*-competitive*, if $E[\text{cost}(A(x))] \leq c \cdot \text{cost}(\text{Opt}(x))$ for every x.

This *competitive analysis* was introduced by Sleator and Tarjan [9] and was, since then, used to analyze many different online problems [5]. To model worst case input instances, we use the idea of an adversary who knows all the algorithm's deterministic decisions and creates the input in a way to harm the algorithm as much as possible. However, the adversary is not able to predict any decisions made based on randomness (in the literature, such an adversary is often called *oblivious*).

Throughout this paper, we consider the following online makespan scheduling problem. Given a factory with m different machines, we are asked to perform a given number of d jobs. Here, a job is a sequence of tasks. Each task asks for one specific machine and for exactly one time unit. In our particular version, every job consists of exactly m tasks, while each task is asking for a different machine. Obviously, in this case, a job can be seen as a permutation of the machines. The jobs must be performed in the given order, the goal is to minimize the completion time, called *makespan*, *i.e.*, the time needed to complete all jobs. As already mentioned, we consider an online setting of the problem, which means that only the current tasks (and all tasks already processed) of the jobs are known to the algorithm which processes the input instance. For an introduction to scheduling problems, see, *e.g.*, [3].

The makespan of a schedule is the sum of the necessary m steps (it is straightforward to see that any solution has to take at least m steps for any instance) and the additional delay *del*. Since the competitive ratio c is defined as the ratio

between the computed solution and the best solution, for the particular problem and for any online algorithm which causes almost a delay of *del*, we conclude that this algorithm has a competitive ratio of

$$c \leq \frac{m + del}{m} = 1 + \frac{del}{m}.$$

Recently, the problem considered in this work has also been investigated in another framework to establish an alternative measurement for the output quality of online algorithms than the competitive ratio [1, 8].

In this paper, we deal with many discrete monotonically increasing functions. We approximate the sums of such a function $f(x)$ by definite integrals. Given that $f(x)$ is strictly increasing, we find, using Riemann's sums that

$$\sum_{x=a}^{b} f(x) \leq \int_{a}^{b+1} f(x)\ \mathrm{d}x.$$

For the ease of presentation, we write c-competitive, instead of $E[c]$-competitive, if it is clear from context that we are talking about a randomized online algorithm. We do not formally talk about expected values of the competitive ratio, but only calculate the average delay over all possible deterministic strategies a randomized algorithm chooses. This equals the competitive ratio, since the constructed randomized algorithms choose all strategies with the same probability. Moreover, we simply identify the function cost with the number of delays that are made.

2. Two jobs with an equal task number – a graphical representation

First, we consider the problem with $d = 2$ jobs both with m tasks. Hromkovič *et al.* [4] found an efficient randomized algorithm for processing these input instances with a delay of at most $\sqrt{m}$. We deal with a modified problem situation and contribute a deterministic algorithm which computes schedules with vanishing delays for certain input instances and we will present a randomized algorithm with a competitive ratio tending to 1 as m increases.

Throughout this paper, we use a graphical representation of the problem as used by Brucker [2]. We use a grid with $m \times m$ cells, thereby creating m columns and m rows. To the axes, we assign a job each, whereby we assign to every column or row a task. The tasks are assigned in the order they are given.

Within a cell, we can graphically represent how tasks are processed. If we process a task, we follow the edge assigned to it. Therefore, processing two tasks simultaneously results in a diagonal edge. If we cannot process two tasks simultaneously, *i.e.*, the tasks require the same machine, we call the diagonal edge *forbidden* and interpret the cell with the conflict as an *obstacle*.

Schedules are represented by a connection between the start corner of the grid and the goal corner. Figure 1 shows the graphical representation of the jobs $J_1 = (1, 2, 3, 4)$ and $J_2 = (1, 3, 2, 4)$ including obstacles and a possible schedule.

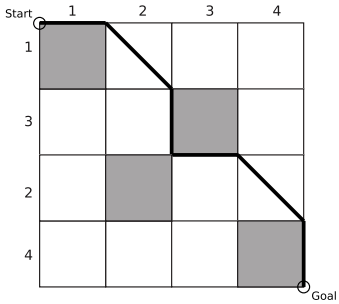
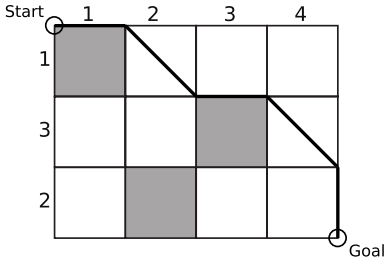
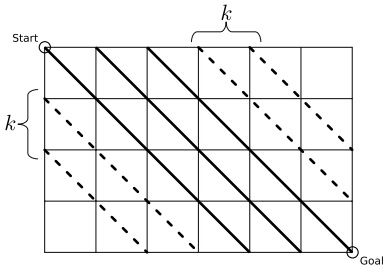


FIGURE 1. Graphical representation of the jobs $J_1 = (1, 2, 3, 4)$ and $J_2 = (1, 3, 2, 4)$ including obstacles and a possible schedule.



(a) Graphical representation of the jobs $J_i = (1, 2, 3, 4)$ and $J_j = (1, 3, 2)$ with $i = 4$ and $j = 3$ including obstacles and a possible schedule.



(b) Grid with shortest and longer schedules, *i.e.*, main and sub-diagonals.

FIGURE 2. Graphical representations of two jobs with unequal task numbers.

3. Two jobs with an unequal task number

Instead of 2 jobs both with m tasks each, we now consider 2 jobs with i and j tasks. We call the job with i tasks J_i and the job with j tasks J_j. The *size* of such an input instance is defined as (i, j). In the following, we assume that $i > j$ and that all tasks contained in J_j are also in J_i. We use the same online setting as before.

We construct a similar graphical representation which consists of a grid with $j \times i$ cells. In Figure 2a, an example, with the jobs $J_i = (1, 2, 3, 4)$ and $J_j = (1, 3, 2)$ with $i = 4$ and $j = 3$ is shown, including obstacles and a possible schedule.

3.1. A deterministic algorithm

We now present a straightforward deterministic algorithm which achieves no delay for any input instance with $i \geq 2j$. The algorithm performs a task of J_i in

every step and, if possible, also a task of J_j is processed. Since there can only be one obstacle per row, the schedule can be delayed in at most every second step. Job J_i is finished after i steps, whereby the schedule has completed at least $\frac{i}{2}$ tasks of J_j. This means that if $j \leq \frac{i}{2} \implies i \geq 2j$, the schedule finishes with no delay at all.

3.2. A randomized algorithm

Next, we give a randomized algorithm that achieves a delay of at most $\sqrt{2j-i}+\frac{1}{2}$ for all input instances with $j < i < 2j$. Since we already know that we do not have any delay if $i \geq 2j$ by using a simple deterministic strategy, we merely look at $j < i < 2j$. Note that, if we neglect the obstacles, there exist multiple shortest ways through the grid. We call diagonals that are part of shortest ways *main diagonals* and diagonals that are part of longer ways *sub-diagonals*. This fact is illustrated in Figure 2b, from where it becomes obvious that a sub-diagonal causes a delay to the schedule equal to its deviation from its closest main diagonal. The algorithm now chooses randomly from the $i-j+1$ main diagonals and additionally from $2\sqrt{2j-i}$ sub-diagonals, half of them below and the other half above the main diagonals[3]. If the algorithm hits an obstacle, it bypasses it by a horizontal and a vertical step.

Theorem 3.1. *The algorithm described above for the* 2*-dimensional case achieves an expected competitive ratio which tends to* 1*.*

Proof. Note that the total delay over all possible diagonals caused by obstacles is at most j. Additionally, sub-diagonals have a delay a which is equal to its deviation from the nearest main diagonal. The sum of all delays is therefore

$$j + 2\sum_{a=1}^{\sqrt{2j-i}} a = j + \sqrt{2j-i}\left(\sqrt{2j-i}+1\right) = \sqrt{2j-i}+3j-i.$$

We can now calculate an upper bound on the expected delay v that the algorithm achieves for an input instance of size (i,j) by

$$\begin{aligned} v &= \frac{\sqrt{2j-i}+3j-i}{i-j+1+2\sqrt{2j-i}} \\ &= \frac{[(i-j)+2\sqrt{2j-i}+1]+(4j-2i-\sqrt{2j-i}-1)}{i-j+1+2\sqrt{2j-i}} \\ &\leq 1+\frac{2(2j-i)-\sqrt{2j-i}-1}{1+1+2\sqrt{2j-i}} \\ &= \frac{1}{2}+\frac{2j-i}{\sqrt{2j-i}+1} \\ &\leq \frac{1}{2}+\sqrt{2j-i}. \end{aligned}$$

[3]The fact that $2\sqrt{2j-i}$ is a good choice within this scope can be verified by an easy calculation.

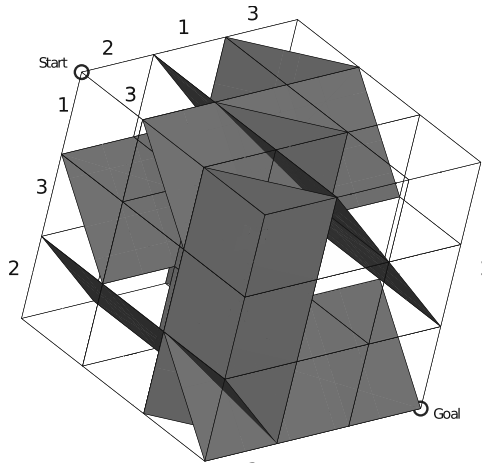


FIGURE 3. 3-dimensional graphical representation of the jobs $J_1 = (2, 1, 3)$, $J_2 = (3, 1, 2)$, $J_3 = (1, 3, 2)$.

Therefore, the expected competitive ratio is at most

$$c \leq 1 + \frac{\frac{1}{2} + \sqrt{2j - i}}{i}$$

which tends to 1 for i, j tending to infinity and $j < i$. □

4. Three jobs with an equal task number

In this section, we consider three jobs with m tasks each. For reasons of clarity, we use a 3-dimensional grid as a graphical representation, where we assign a job to every axis. We obtain a cube with $m \times m \times m$ cells, where, as above, following an edge within such a cell corresponds to processing the associated task of this edge. Again, processing multiple jobs in the same time unit results in a diagonal edge.

Every pair of jobs, which ask for the same machine, leads to collisions of requests. We define an obstacle as such a collision of requests between two jobs. This fact is illustrated in Figure 3. Since there are 3 pairs of jobs, there are $3m$ obstacles. Obstacles are shown as areas, where all diagonal edges contained within such an area are called *forbidden*. As for the case with two machines, schedules are represented by a connection between the start and the goal corner, where the number of edges contained in a particular schedule corresponds to the makespan of this schedule. In one time unit, it is possible to process at most 3 tasks, given that they are all distinct. Processing only 2 or less tasks in one time unit causes a delay to the schedule.

Again, we look at the grid without obstacles. As above, we call the (in this case unique) shortest way through the grid the main diagonal and longer ways parallel to this main diagonal are called sub-diagonals. These sub-diagonals start in one of the side planes of the cube and end in another side plane. The point in the side plane, where a sub-diagonal starts, is called the *displacement* of this sub-diagonal. Moving to the starting point (i, j) of a sub-diagonal leads to a 2-dimensional subproblem. By (i, j) we denote the size of this subproblem.

4.1. Randomized algorithm

Based on the algorithm found in Sections 3.1 and 3.2, and the algorithm presented in [4], we create an algorithm for processing problem instances with 3 jobs. We apply the makespans of the following classes of problem instances.

(1) Hromkovič *et al.* [4] found an algorithm for processing instances with 2 jobs of size (m, m) and showed an upper bound for the delay of $\sqrt{m}$;
(2) in Section 3.1, we found an algorithm with a delay of 0 for problem instances of size (i, j), where $i \geq 2j$;
(3) in Section 3.2, we found an algorithm for 2 jobs of size (i, j). We showed an upper bound of $\sqrt{2j - i} + \frac{1}{2}$ for the delay of this algorithm.

We now create a set of deterministic strategies Δ and choose one of them randomly. This set contains sub-diagonals (close to the main diagonal) and the main diagonal. Our algorithm first moves to the starting point of the randomly chosen diagonal, then approximates this diagonal and eventually finishes the jobs when the side plane of the cube is reached, *i.e.*, one of the three jobs is finished. We define the algorithm as follows:

Step 1. Choose a point randomly. The point is of the form (s_1, s_2, s_3) , where one coordinate is 0 and the other two have any value between and including 0 and r where $r \in \mathbb{N}$, $r \leq m$.

Step 2. Move to the chosen point by using the algorithm we have presented in Section 3.

Step 3. Follow the diagonal, parallel to the main diagonal, starting from the chosen point. Bypass any obstacle by processing the tasks in conflict one after the other and let the task not involved in the conflict (if there is any) process while the others are processing.

Step 4. As soon as one job is finished, complete the last two jobs by using the algorithm presented in Section 3.

Theorem 4.1. *The algorithm developed for the* 3*-dimensional case has an expected competitive ratio that tends to* 1 *as* m *tends to infinity for* $r = O(\sqrt(m))$ *and features a delay that tends to approximately* $2.309\sqrt{m}$.

Proof. For the sake of readability, we divide the analysis into different parts.

Deterministic strategies and their geometric properties.

In order to minimize the delay, we choose sub-diagonals which are as close as possible to the main diagonal. In order to reach the starting point of a sub-diagonal, we face a *first* 2*-dimensional subproblem* of size (i, j). Without loss of generality, we assume that $i \geq j$ and we define J_i as the job on which we process i tasks and J_j as the job on which we process j tasks. The job on which no tasks are processed in this subproblem is called J_0.

After dealing with this subproblem, we approximate the diagonal. We call this subproblem the 3*-dimensional subproblem*. With every step along the diagonal, we process exactly one task on all jobs. The diagonal has a length of exactly $m - i$, since then J_i is finished, meaning that a side plane of the cube next to the goal corner is reached.

Again, we face a 2-dimensional subproblem. So far, we have processed $m - i$ tasks on J_0 and $m - i + j$ tasks on J_j. This leads to the *second* 2*-dimensional subproblem* of size $(i, i - j)$.

If we do not consider the obstacles that were met, we find an makespan of $i + m - i + i = m + i$, meaning that a delay of i is caused to the whole schedule.

We now sum up the delays over all strategies. We divide the set of the starting points into 3 side planes P_{1-3} and 3 edges E_{1-3}. All 3 edges have the same length and all the planes have the same size. Therefore, the summed delay caused by the diagonals starting in them is also the same.

Recall that the coordinates of the starting points are of the form (s_1, s_2, s_3), where one coordinate is 0 and the other two are $i, j \in \{0, \ldots, r\}$. As already mentioned, and without losing generality, we assume that $i \geq j$.

Number of strategies.

The number of strategies is given by the number of starting points. These points lie within three edges with length r and three areas of size (r, r) and the start corner. Each area leads to r^2 starting points and each edge to r starting points. The number of diagonal strategies is therefore given by

$$3r^2 + 3r + 1.$$

Delay caused by the deviation from main-diagonal.

First, we sum up the delay over all possible starting points which lie on one distinct edge yielding

$$D_E = \sum_{i=1}^{r} i = \frac{r(r+1)}{2}.$$

Next, we sum up the delay over all the starting points which are on the plane P. Note that, for every i, we have $2i - 1$ possible starting points on P each of which causes a delay of i. Therefore, we have

$$D_P = \sum_{i=1}^{r} (2i^2 - i) = 2\sum_{i=1}^{r} i^2 - \sum_{i=1}^{r} i = \frac{2r^3}{3} + \frac{r^2}{2} - \frac{r}{6}.$$

Since there are 3 planes and 3 edges, we count a delay of

$$D_{\text{dev}} = 3D_P + 3D_E = 2r^3 + 3r^2 + r.$$

Delay caused by obstacles in the first 2-dimensional subproblem.

The delay caused by obstacles is obviously 0 for starting points on any of the edges. As for the planes, we divide a plane P into 3 regions.

Region 1 (where $i = j$). We use the algorithm presented by Hromkovič *et al.* [4] which achieves a delay of at most $\sqrt{i} = \sqrt{j}$.

Region 2 (where $i \geq 2j$). We use the deterministic algorithm presented in Section 3.1 which achieves a delay of 0.

Region 3 (where $j < i < 2j$). We use the randomized algorithm found in Section 3.2 which leads to an expected delay of at most $\sqrt{2j - i} + \frac{1}{2}$.

In what follows, we look at the delays of any region in detail.

Delay in Region 1. We easily approximate the sum as

$$\begin{aligned}\sum_{i=1}^{r} \sqrt{i} &\leq \int_1^{r+1} \sqrt{i}\, \mathrm{d}i \\ &= \frac{2}{3}\left((r+1)^{\frac{3}{2}} - 1\right) \\ &\leq \frac{2}{3}(r+1)^{\frac{3}{2}}.\end{aligned}$$

Since there are 3 such regions, we need to multiply this number by 3.

Delay in Region 2. The delay in this region is obviously 0 as all the schedules can reach their starting points without any delay.

Delay in Region 3. Again, we sum up the delays for $i = 1$ to r, where in every step $\frac{i}{2} < j < i$. Since we are using natural numbers, this even simplifies to $\frac{i}{2} < j \leq i-1$. As we have shown in Section 3.2, the delay for a specific pair i, j is $\sqrt{2j - i} + \frac{1}{2}$ and so we get

$$\begin{aligned}&\sum_{i=1}^{r} \sum_{j=\lfloor \frac{i}{2}+1 \rfloor}^{i-1} \left(\sqrt{2j-i} + \frac{1}{2}\right) \leq \sum_{i=1}^{r} \left(\int_{\frac{i}{2}}^{i} \sqrt{2j-i} + \frac{1}{2}\, \mathrm{d}j\right) \\ &= \sum_{i=1}^{r} \left(\left[\frac{1}{3}(2j-i)^{\frac{3}{2}} + \frac{j}{2}\right]_{\frac{i}{2}}^{i}\right) \quad = \quad \sum_{i=1}^{r} \left(\frac{1}{3}i^{\frac{3}{2}} + \frac{i}{4}\right) \\ &= \frac{1}{3}\sum_{i=1}^{r} i^{\frac{3}{2}} + \frac{1}{4}\sum_{r=1}^{r} i \qquad\qquad \leq \frac{1}{3}\int_1^{r+1} i^{\frac{3}{2}}\, \mathrm{d}i + \frac{r(r+1)}{8} \\ &\leq \frac{2}{15}(r+1)^{\frac{5}{2}} + \frac{1}{8}r^2 + \frac{1}{8}r.\end{aligned}$$

We need to multiply this result by 3 for every plane and by 2 for every region, yielding a delay of at most

$$\frac{4}{5}(r+1)^{\frac{5}{2}} + \frac{3}{4}r^2 + 2(r+1)^{\frac{3}{2}} + \frac{3}{4}r.$$

Delay caused by obstacles in the second 2-dimensional subproblem.

Here, we deal with the same type of problem as in Section 4.1. Instead of a problem of size (i, j), we meet an input instance of size $(i, i-j)$. There is only a delay, if $2(i-j) > i \implies i \geq 2j \implies \frac{i}{2} \geq j$. This is exactly the complementary of the case we investigated above which leads to the same outcome.

Delay caused by obstacles in the 3-dimensional subproblem.

We now prove the following lemma.

Lemma 4.1. *A single obstacle affects at most* $2r+1$ *diagonals in* Δ.

Proof. As a first step, we parameterize the diagonals with $a, b \in \{0, \ldots, r\}$, as the components of the starting point. All diagonals are parallel to the main diagonal, and are thereby parallel to $(1,1,1)^T$. Moreover, the variable t denotes the position within the diagonal. The different classes of starting points yield three classes of diagonals which are the following.

$$\overrightarrow{r} = \begin{pmatrix}1\\1\\1\end{pmatrix} t + \begin{pmatrix}a\\b\\0\end{pmatrix}.$$

Diagonals with starting points of the form $(a, b, 0)$.

$$\overrightarrow{r} = \begin{pmatrix}1\\1\\1\end{pmatrix} t + \begin{pmatrix}a\\0\\b\end{pmatrix}.$$

Diagonals with starting points of the form $(a, 0, b)$.

$$\overrightarrow{r} = \begin{pmatrix}1\\1\\1\end{pmatrix} t + \begin{pmatrix}0\\a\\b\end{pmatrix}.$$

Diagonals with starting points of the form $(0, a, b)$.

In order to count the intersections of these diagonals with an obstacle, we also parameterize the obstacles. Essentially, there are three types of obstacles, namely one for every pair of jobs. However, since all classes of obstacles are symmetric, it is sufficient to consider obstacles of the type $b = (k, l, z)$, with k, l denoting the position of the obstacle in the grid and z being arbitrary between 0 and m. We need to find k, l, such that the number of intersections with diagonals of all three classes are maximal.

For all three cases, we find the intersections with the diagonals by equating the parameterized equations of the obstacles with the diagonals. Eliminating t yields equalities for a, b in dependence of k, l. By definition, it holds for a, b that

$0 \leq a, b \leq r$. For k, l holds $0 \leq k, l \leq m$, since an obstacle can be positioned anywhere in the grid.

$$\begin{pmatrix}1\\1\\1\end{pmatrix} t + \begin{pmatrix}a\\b\\0\end{pmatrix} = \begin{pmatrix}k\\l\\z\end{pmatrix} \qquad \begin{pmatrix}1\\1\\1\end{pmatrix} t + \begin{pmatrix}a\\0\\b\end{pmatrix} = \begin{pmatrix}k\\l\\z\end{pmatrix} \qquad \begin{pmatrix}1\\1\\1\end{pmatrix} t + \begin{pmatrix}0\\a\\b\end{pmatrix} = \begin{pmatrix}k\\l\\z\end{pmatrix}$$

which gives us

$$\begin{aligned} a + t &= k \\ b + t &= l \\ t &= z \\ \Longrightarrow\ k - a &= l - b, \end{aligned}$$

which gives us

$$\begin{aligned} a + t &= k \\ t &= l \\ b + t &= z \\ \Longrightarrow\ a + l &= k, \text{ and} \end{aligned}$$

which gives us

$$\begin{aligned} t &= k \\ a + t &= l \\ b + t &= z \\ \Longrightarrow\ a + k &= l. \end{aligned}$$

This leads to the following three equalities.

$$a - b = k - l \tag{4.1}$$

$$a = k - l \tag{4.2}$$

$$a = l - k. \tag{4.3}$$

We continue to analyze these equalities, which describe the number of intersections possible. We count the number of diagonals that intersect an obstacle. The position of the diagonal is determined by a, b, whereby the position of the obstacle only depends on k, l. In order to simplify the analysis, we distinguish between the case where $k - l \geq 0$ and the case where $k - l < 0$. Both cases are mutually exclusive and the maximal number of intersections of both cases determines the result.

Case 1. With equation (4.1), we obtain at most $r + 1$ intersections, since there exist exactly $r+1$ choices of a, b with $a-b = k-l$. The maximal number of choices is fulfilled if $k - l = 0$, since in these cases $a = b$, which leaves $r + 1$ choices for a and b open. If $k - l = c$ for $c \in \{0, \ldots, r\}$, we find $a = c + b$, which leaves $r + 1 - c$ choices for a, b open, *i.e.*, $r + 1$ is the maximum.

Equation (4.2) states that $a = k - l$, *i.e.*, a has a fixed value, since $k - l$ is constant. However, the number of intersection does not depend on b, which means that it can take any value that is allowed for it. This leaves exactly $r + 1$ choices open.

Equation (4.3) cannot contribute any intersection, since we consider $k - l \geq 0$ and $a \geq 0$. This leaves us so far $2r+2$ intersections. However, we have counted the starting point $(0, 0, 0)$ twice, since by considering equation (4.1) we count $a = b = 0$. Equation (4.2) was independent of b, which means that b is once 0. If $k - l = 0$, which is the case for obtaining the maximal possible number of intersections with equation (4.1), we count $(0, 0, 0)$ again. The number of intersections for this case is thus $2r + 1$.

Case 2. We reformulate equation (4.1) to $b-a = l-k$. We obtain a symmetric case in comparison to the first one which means that not more than $2r+1$ intersections are possible.

This proves our Lemma 4.1. □

Note that, since there are exactly $3m$ obstacles, there are at most $3m(2r+1)$ delays possible.

Average delay.

We are now ready to calculate the average delay over all strategies that might be chosen by our algorithm. To sum up, we found the following terms.

- The delay in any of the first and second 2-dimensional subproblem caused by obstacles is at most
$$\frac{4}{5}(r+1)^{\frac{5}{2}} + \frac{3}{4}r^2 + 2(r+1)^{\frac{3}{2}} + \frac{3}{4}r;$$
- the delay in the 3-dimensional subproblem caused by obstacles is at most
$$3m(2r+1);$$
- the delay caused by the deviation of the strategies from the main diagonal is
$$2r^3 + 3r^2 + r;$$
- the number of diagonal strategies is
$$3r^2 + 3r + 1.$$

This enables us to find the expected delay over all strategies. We get

$$v(r) \leq \frac{2r^3 + \frac{8}{5}(r+1)^{\frac{5}{2}} + \frac{9}{2}r^2 + 4(r+1)^{\frac{3}{2}} + (6m + \frac{5}{2})r + 3m}{3r^2 + 3r + 1},$$

which leads to a competitive ratio of

$$c \leq 1 + \frac{2r^3 + \frac{8}{5}(r+1)^{\frac{5}{2}} + \frac{9}{2}r^2 + 4(r+1)^{\frac{3}{2}} + (6m + \frac{5}{2})r + 3m}{m(3r^2 + 3r + 1)}.$$

It is thus immediately clear that, if we choose $r = O(\sqrt{m})$, the competitive ratio tends to 1 as m tends to infinity. We determine the delay numerically by increasing the number of tasks m and we calculate the minimum of the average delay, which tends to approximately $2.309\sqrt{m}$. This is illustrated in Figure 4. □

5. Conclusion

We optimized a randomized algorithm presented by Hromkovič *et al.* [4] and we gave an in-depth analysis of the base case of the problem. Furthermore, we created a randomized algorithm for 2 jobs with a different number of tasks. This algorithm has also been applied to processing problem instances with 3 jobs.

All the considered randomized algorithms have the property that they have a competitive ratio tending to 1 as the number of tasks increases. This is, however,

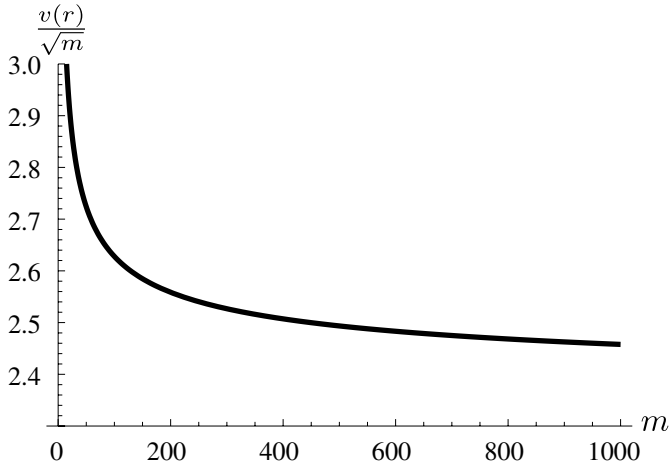


FIGURE 4. Asymptotic behavior of the minimum of $\frac{v(r)}{\sqrt{m}}$ as m tends to infinity.

nothing new so far, as the algorithms presented in [4] already have this property. However, in practical applications not only the competitive ratio is relevant, but also the overall completion time. We significantly reduced the overall completion time of processing problem instances with 3 jobs to approximately $m + 2.309\sqrt{m}$, which improves the result found by [4] of $m+6\sqrt{m}$ using an algorithm for a general number of jobs d.

All in all, we showed several improvements for algorithms for processing specific types of instances of the problem and we created algorithms for a specific version of the problem which has not been considered before.

A next step could be an analysis of the problem in general, with an arbitrary number of jobs, each with an arbitrary number of tasks. It remains yet unclear whether the techniques used to find and improve the algorithms can be generalized in this form.

Acknowledgements. The authors would like to thank Juraj Hromkovič, Karin Freiermuth, and Dennis Komm for their support with ideas, proof-reading, and corrections.

REFERENCES

[1] H.-J. Böckenhauer, D. Komm, R. Královič, R. Královič and T. Mömke, On the advice complexity of online problems, in *Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC 2009). Lect. Notes Comput. Sci.* **5878** (2009) 331–340.

[2] P. Brucker, An efficient algorithm for the job-shop problem with two jobs. *Computing* **40** (1988) 353–359.

[3] P. Brucker, *Scheduling Algorithms*, 4th edition. Springer-Verlag (2004).

[4] J. Hromkovič, T. Mömke, K. Steinhöfel and P. Widmayer, Job shop scheduling with unit length tasks: bounds and algorithms. *Algorithmic Oper. Res.* **2** (2007) 1–14.
[5] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press (1998).
[6] J. Hromkovič, *Design and Analysis of Randomized Algorithms*. Springer-Verlag (2006).
[7] S. Irani and A.R. Karlin, On online computation, in *Approximation Algorithms for NP-hard Problems, Chapter 13*, edited by Hochbaum. PWS Publishing Company (1997) 521–564.
[8] D. Komm and R. Kálovič, Advice complexity and barely random algorithms. *Theoret. Inform. Appl.* **45** (2011) 249–267.
[9] D.D. Sleator and R.E. Tarjan, Amortized efficiency of list update and paging rules. *Commun. ACM* **28** (1985) 202–208.

Communicated by J. Hromkovič.
Received September 1st, 2010. Accepted November 3rd, 2011.