Algorithmic Operations Research

Job Shop Scheduling with Unit Length Tasks: Bounds and Algorithms

Juraj Hromkovič, Tobias Mömke, Kathleen Steinhöfel et Peter Widmayer

Volume 2, numéro 1, summer 2007

URI : https://id.erudit.org/iderudit/aor2_1art01

Aller au sommaire du numéro

Éditeur(s)

Preeminent Academic Facets Inc.

ISSN

1718-3235 (numérique)

Découvrir la revue

Citer cet article

Hromkovič, J., Mömke, T., Steinhöfel, K. & Widmayer, P. (2007). Job Shop Scheduling with Unit Length Tasks: Bounds and Algorithms. *Algorithmic Operations Research*, *2*(1), 1–14.

Résumé de l'article

We consider the job shop scheduling problem unit–Jm, where each job is processed once on each of m given machines. Every job consists of a permutation of tasks for all machines. The execution of any task on its corresponding machine takes exactly one time unit. The objective is to minimize the overall completion time, called makespan. The contribution of this paper are the following results: (i) For any input instance of unit–Jm with d jobs, the makespan of an optimum schedule is at most m + o(m), for d = o(m1/2). This improves on the upper bound O(m + d) given in [5] where O hides a constant equal to two as shown in [7]. For d = 2 the upper bound is improved to m + ⌈ √m ⌉. (ii) There exist input instances of unit–Jm with d = 2 such that the makespan of an optimum schedule is at least m + ⌈ √m ⌉, i.e., our upper bound for d = 2, see result (i), cannot be improved. (iii) We present a randomized on-line approximation algorithm for unit–Jm with the best known approximation ratio for d = o(m1/2). (iv) There is no deterministic on-line algorithm with a competitive ratio better than 4/3 for unit–Jm with two jobs, and for three or more jobs, there is no deterministic on-line algorithm which is better than 1.5 competitive. Compared with the expected competitive ratio of (iii) which tends to 1, this shows that for unit–Jm randomization is very powerful compared with determinism. For two and three jobs, deterministic on-line algorithms with competitive ratios tending to 4/3 and 1.5 respectively are presented. (v) A deterministic approximation algorithm for unit–Jm is described that works in quadratic time for constant d and has an approximation ratio of 1 + 2d/⌊ √m ⌋ for d ⩽ 2 log2 m.

Algorithmic Operations Research Vol.2 (2007) 1–14

Job Shop Scheduling with Unit Length Tasks: Bounds and Algorithms

Juraj Hromkovič [a] Tobias Mömke [a] Kathleen Steinhöfel [b] Peter Widmayer [a]

[a]Department of Informatics, ETH Zurich, ETH Zentrum, CH-8092 Zürich, Switzerland.
[b]King's College London, Department of Computer Science, Strand, WC2R 2LS London, UK
and FIRST – Fraunhofer Institut für Rechnerarchitektur und Softwaretechnik, Kekuléstraße 7, 12489 Berlin, Germany.

Abstract

We consider the job shop scheduling problem unit–J_m, where each job is processed once on each of m given machines. Every job consists of a permutation of tasks for all machines. The execution of any task on its corresponding machine takes exactly one time unit. The objective is to minimize the overall completion time, called makespan. The contribution of this paper are the following results: (i) For any input instance of unit–J_m with d jobs, the makespan of an optimum schedule is at most $m + o(m)$, for $d = o(m^{1/2})$. This improves on the upper bound $O(m + d)$ given in [5] where O hides a constant equal to two as shown in [7]. For $d = 2$ the upper bound is improved to $m + \lceil \sqrt{m} \rceil$. (ii) There exist input instances of unit–J_m with $d = 2$ such that the makespan of an optimum schedule is at least $m + \lceil \sqrt{m} \rceil$, i.e., our upper bound for $d = 2$, see result (i), cannot be improved. (iii) We present a randomized on-line approximation algorithm for unit–J_m with the best known approximation ratio for $d = o(m^{1/2})$. (iv) There is no deterministic on-line algorithm with a competitive ratio better than $4/3$ for unit–J_m with two jobs, and for three or more jobs, there is no deterministic on-line algorithm which is better than 1.5 competitive. Compared with the expected competitive ratio of (iii) which tends to 1, this shows that for unit–J_m randomization is very powerful compared with determinism. For two and three jobs, deterministic on-line algorithms with competitive ratios tending to $4/3$ and 1.5 respectively are presented. (v) A deterministic approximation algorithm for unit–J_m is described that works in quadratic time for constant d and has an approximation ratio of $1 + 2^d/\lfloor\sqrt{m}\rfloor$ for $d \leq 2\log_2 m$.

Key words: Scheduling, Makespan, Multiple machines, Approximation Algorithms, Online Algorithms, Competitive ratio

1. Introduction

Minimizing the makespan for general job shop scheduling is one of the fundamental optimization problems. It is NP-hard, and Williamson et al. [9] proved that the minimum makespan is not even approximable in polynomial time within $5/4 - \varepsilon$ for any ε. Moreover, no constant approximation algorithm is known, see Goldberg et al. [3] and Shmoys et al. [8].

Many job shop scheduling models have been identified as having a number of practical applications. But even severely restricted models remain strongly NP-hard. In this paper, we consider a problem setting that relates to finding optimum schedules for routing packets through a network, see [5]. It is a well-studied version of job shop scheduling with m different machines and unit length tasks, denoted by *unit–J_m*. There are d jobs $J_1, J_2, \ldots, J_d$ for some integer $d \geq 2$. Each job consists of a sequence of m tasks, such that each machine processes exactly one task of the job. Therefore, for each job the order of the tasks $\sigma_1, \sigma_2, \ldots, \sigma_m$ determines a permutation of the m machines, where σ_i requires processing on the i-th machine. As in the general job shop, each machine can process only one task at a time and each job must be executed on the machines in the order given by its permutation. A feasible schedule is an assignment of starting times to tasks that satisfies all stated restrictions. The makespan of a schedule is the maximum over the completion times of all jobs. The objective is to minimize the makespan over all feasible schedules. The problem *unit–J_m* is NP-hard for $m \geq 3$, see Lenstra and Rinnooy Kan [6].

The algorithm of Goldberg et al. [3] improved a result of Shmoys et al. [8] and provides an approximation ratio $O((\log_2 m)/(\log_2 \log_2 m)^2)$ for *unit–J_m*. Instances with two jobs have been shown by Brucker [1] to be solvable in linear time. Later, we shall see that a straightforward extension of this algorithm leads to an $O(m^d)$ time algorithm for any input instance of *unit–J_m* with d jobs. Leighton et al. [4,5] proved that there exists always a schedule with makespan $O(m + d)$. This pro-

⋆ Supported in part by SNF grant 200021-107327/1

vides a randomized constant approximation algorithm for this problem. The constant is equal to two and was determined by Scheideler [7]. Feige and Scheideler [2] proved that the bound does not extend to the case of arbitrary task lengths.

In this paper, we analyze the hardest input instances of *unit–*J_m. As already mentioned, finding the optimal makespan of job shop instances with two jobs is solvable in linear time. Therefore, in this paper, the term hard instance is used in the sense of makespan length only. Our observations lead to the design of a randomized on-line algorithm that solves *unit–*J_m with d jobs in linear time with expected approximation ratio that tends to 1 for $d = o(m^{1/2})$. The contributions of this paper can be formulated as follows.

(1) The makespan of an optimum schedule is at most

$$m + 2d\sqrt{m};$$

this amounts to $m + o(m)$ for every problem instance of *unit–*J_m with $d = o(m^{1/2})$ jobs, and thus, for this case, improves on the upper bound $O(m + d)$ derived by Leighton et al. [5], where O hides a constant of two as shown in [7]. For $d = 2$ we prove the stronger upper bound $m + \lceil\sqrt{m}\rceil$.

(2) There exist input instances of *unit–*J_m with two jobs such that every schedule has a makespan of at least

$$m + \sqrt{m}.$$

Hence, the result for $d = 2$, see (i), cannot be improved.

(3) For every positive integer m, there is a randomized on-line approximation algorithm that solves *unit–*J_m in linear time with an expected approximation ratio of

$$1 + \frac{2d}{\sqrt{m}};$$

this amounts to $1+o(1)$ for $d = o(m^{1/2})$. These results demonstrate an extreme power of randomness for *unit–*J_m for several reasons. First of all our randomized on-line algorithm is competitive with respect to the makespan of an optimum solution. For $d = o(m^{1/2})$ the algorithm is the best approximation algorithm for *unit–*J_m. We do not know any off-line polynomial-time approximation algorithm with an approximation ratio that would tend to 1 for $d = o(m^{1/2})$ with growing m. Moreover, no deterministic on-line algorithm can achieve a makespan better than $d \cdot (m-1)/\log_2 d$ [5].

(4) For every on-line algorithm for *unit–*J_m, we present an input instance with $d = 2$ such that the algorithm results in a makespan of at least $m + m/3$. We also present a deterministic 4/3-competitive on-line algorithm for input instances of *unit–*J_m with $d = 2$. Similarly, for every deterministic on-line algorithm with $d = 3$, we present an input instance such that the algorithm results in a makespan of at least $m + m/2$ and we present a deterministic on-line algorithm that has a competitive ratio tending to 1.5 with m growing for input instances of *unit–*J_m with $d = 3$. We show that there is no deterministic on-line algorithm with $d = 2$ and a competitive ratio better than $4/3$, and for $d \geq 3$, the competitive ratio of every deterministic on-line algorithm is at least 1.5. This is significantly worse than the expected competitive ratio tending to 1 of the randomized on-line algorithm of (iii). This clearly shows the power of randomization for the on-line version of *unit–*J_m.

(5) We present a deterministic approximation algorithm that is efficient at least for small d's in comparison with m. Its run-time is $O(d^2m^2)$, and it has an approximation ratio of at most

$$1 + \frac{2^d}{\lfloor\sqrt{m}\,\rfloor}$$

which tends to 1 with growing m for $d = o(\log_2 m)$.

The paper is organized as follows. Section 2. presents a geometrical representation of the input instances of *unit–*J_m that is essential for a transparent analysis of *unit–*J_m. In Section 3. we present some hard input instances with two jobs only. Section 4. shows the existence of efficient schedules for all input instances of *unit–*J_m. In Section 5. the randomized algorithm with the properties as described in (iii) is given. In Section 6. we give hard instances and algorithms for on-line *unit–*J_m and compare them with the previous randomized results. Our deterministic approximation algorithm is presented in Section 7.

2. A geometrical representation of instances

We start with the representation of input instances with two jobs that was employed in [1] to design a linear time algorithm for this special case of *unit–*J_m.

Let $(i_1, \ldots, i_m)$ and $(j_1, \ldots, j_m)$ be two permutations of $(1, 2, \ldots, m)$ that represent the input instance

$(\sigma_{i_1}, \sigma_{i_2}, \ \ldots\ , \sigma_{i_m})$, $(\sigma_{j_1}, \sigma_{j_2}, \ \ldots\ , \sigma_{j_m})$ of *unit–J_m*. We consider a grid G_m of size $m \times m$, where for all $k, l \in \{1, \ \ldots\ , m\}$ the k-th row of G_m is labeled by j_k and the l-th column of G_m by i_l. A pair (k, l), i.e., the intersection of the k-th row and the l-th column, is called an **obstacle**, if and only if $i_l = j_k$. The corresponding square is depicted by a black box.

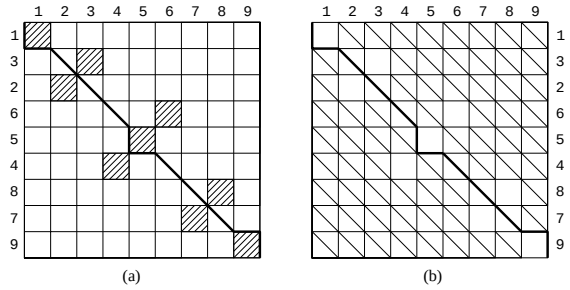

Fig. 1. A hard input instance of *unit–J_m* with two jobs and nine machines.

Fig. 1a illustrates the G_9 of the input instance with two jobs that are given by the two permutations $(1, 2, 3, 4, 5, 6, 7, 8, 9)$ and $(1, 3, 2, 6, 5, 4, 8, 7, 9)$. The term obstacle is motivated by the following observation. Assume that the first job has executed its first $l - 1$ tasks and the second job its first $k - 1$ tasks. If $i_l = j_k$, then both tasks σ_{i_l} and σ_{j_k} require the same machine and therefore, only one of the two jobs can continue its execution in the next time unit and the other one is **delayed**. Otherwise, both jobs can proceed simultaneously.

We assign to the grid G_m the Graph(G_m)=(V, E), where V consists of all vertices of the grid and the set E includes all orthogonal edges of the grid. Additionally, E contains diagonal edges that connect the upper-left corner with the lower-right corner of a grid square that is not an obstacle. Fig. 1b shows the corresponding Graph(G_9) of G_9 given in Fig. 1a. Any feasible schedule is represented by a path from the upper-left corner of G_9 to the lower-right corner of G_9. The path consists of edges of Graph(G_9), where each edge represents one unit of time. A vertical grid edge indicates that in this time unit, a task of the first job is **delayed**; a horizontal grid edge indicates a **delay** of a task of the second job; a diagonal edge tells that both jobs are processed at the same time with no delay.

An optimum schedule corresponds to a shortest path from the upper-left corner a to the lower-right corner b. The bold polygonal line in Fig. 1 represents an optimum schedule of our example. In the schedule, there are 6 delays that are equally distributed between the two jobs. Therefore, the makespan of the illustrated schedule is $m + 6/2 = 9 + 3 = 12$.

Let S be a schedule of an instance with two jobs. The number of vertical edges of the path representing S is called the *delay of the first job according to S*, and the number of horizontal edges of S is called the *delay of the second job according to S*. The **delay of S** is the maximum over these two delays. Obviously, the makespan of S is exactly the sum of m and the delay of S. For later use, we denote by **sum-delay(S)** the sum of the delays of jobs according to S.

We outline the extension of this representation for an arbitrary number d of jobs. In this case we have a d-dimensional grid G_m^d that contains m^d d-dimensional grid cells. Again, the unit intervals of each axis are labeled by the tasks according to the sequence of machines of the corresponding job. Fixing a label i of some axis results in a $(d-1)$-dimensional subgrid of G_m^d.

The intersection of two such different subgrids with labels i is a $(d-2)$-dimensional subgrid of m^{d-2} grid hypercubes that are obstacles in the following sense. Let d' and d'' be the subgrids resulting from a common label i on two axes a' and a''. Any diagonal of a grid square Q in the intersection of d' and d'' whose projection in the two dimensional subspace determined by a' and a'' is a diagonal corresponds to the execution of 2 tasks on the same machine. Therefore, any such diagonal cannot be part of a path determining the makespan and will be called *forbidden*. All other diagonals are allowed w.r.t this intersection. In particular, the main diagonal of such a square Q (that corresponds to the execution of all tasks determined by the coordinates of this grid square Q) is forbidden, and so are the diagonals on the surface of Q that is defined by a' and a''. For instance, if Q is part of the intersection of q $(d-1)$-dimensional subgrids determined by the same label i on q different axes, then to go from the "lowest" corner of Q to the opposite corner of Q, requires at least q time units: Since in this case q tasks request the same machine, this congestion can be resolved by q subsequent steps only. Fig. 2 gives an example of such an obstacle in the 3-dimensional case.

Again, any optimum schedule corresponds to a shortest path between the two extreme corners of the grid. Therefore, for any constant d we get a polynomial-time algorithm for input instances with d jobs. The notions

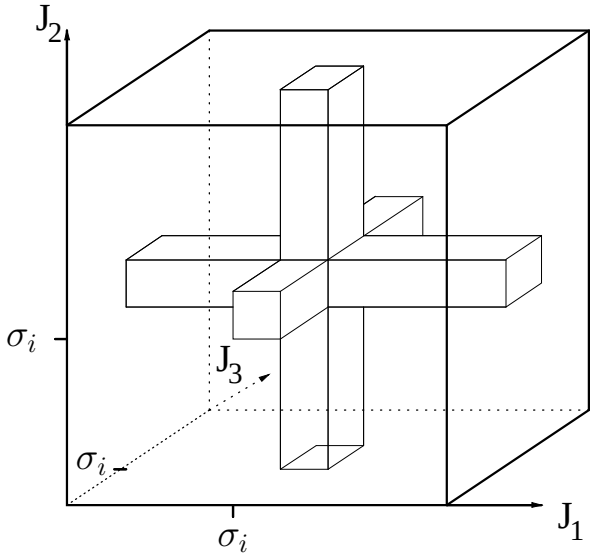

Fig. 2. An obstacle in the 3-dimensional case.

delay of S and *sum-delay(S)* can be extended for $d > 0$ jobs in a straightforward way.

3. Some hard instances

The aim of this section is to construct some of the hardest problem instances with two jobs, i.e., instances where the optimum schedule has a maximum number of delays. Let **makespan(I)** denote the length of an optimum schedule for the problem instance I in what follows.

Lemma 1 *For every* $m = \binom{k+1}{2}, k$ *a positive integer, there exists an input instance* I_R *of two job unit–*J_m *such that*

$$makespan(I_R) \geq m + \sqrt{\frac{m}{2}} - \frac{1}{2}.$$

Proof: Let $I_R^m = (J_1, J_2)$, where

$$J_1 = w_1, w_2, \ \ldots\ , w_k\ , \text{ and } J_2 = w_1^R, w_2^R, \ \ldots\ , w_k^R,$$

with w_i denoting a subsequence of machine indices induced by the tasks, and w_i^R denoting the reverse of w_i. The subsequences w_i and w_i^R, with $i = 1, \ \ldots\ , k$, are defined as

$$w_i = [\tbinom{i}{2} + 1, \tbinom{i}{2} + 2, \ \ldots\ , \tbinom{i}{2} + i - 1, \tbinom{i}{2} + i]$$

and hence

$$w_i^R = [\tbinom{i}{2} + i, \tbinom{i}{2} + i - 1, \ \ldots\ , \tbinom{i}{2} + 2, \tbinom{i}{2} + 1].$$

Observe that w_i is a sequence of i integers, for $i = 1, \ \ldots\ , k$, and that $J_1 = 1, 2, \ \ldots\ , m$. An example for $m = 10 = \binom{5}{2}$ is

$$J_1 = [1], [2, 3], [4, 5, 6], [7, 8, 9, 10],$$

and

$$J_2 = [1], [3, 2], [6, 5, 4], [10, 9, 8, 7].$$

Clearly, if no delay occurs among the two jobs the makespan would be m. Therefore, we show that every schedule on I_R^m contains at least k delays, i.e., every shortest path contains at least k orthogonal grid edges. Every of these orthogonal edges delays either of the two jobs. Therefore, we have at least an overall delay of $k/2$, i.e., the makespan must be at least $m + k/2$. Note that it is sufficient to prove that $k/2 \geq \sqrt{m}/\sqrt{2} - 1/2$ because it implies $makespan(I_R) \geq m + \sqrt{m/2} - 1/2$. We prove by induction on $i + 1$ that any schedule for the jobs $(w_1, w_2, \ \ldots\ , w_i)$ and $(w_1^R, w_2^R, \ \ldots\ , w_i^R)$ causes at least i delays. To do so, we use the following induction hypothesis:

Any schedule for $\overline{I}_R^{\binom{i+1}{2}}$*, where one job is completed and for the other job a prefix of length* $\binom{i+1}{2} - r$*, for* $r \leq i$*, is already processed (*r *is called the* relative delay*), uses at least* i *orthogonal grid edges (sum-delay is at least* i*), and it uses at least* $i + 1$ *orthogonal grid edges if the parities of* r *and* i *differ (i.e.,* r *is odd and* i *is even, or* r *is even and* i *is odd).*

Obviously, this is true for $i = 1$. Let the hypothesis be true for $i' = i - 1$.

Now, consider a prefix of a schedule S for $\overline{I}_R^{\binom{i+1}{2}}$, $i > 1$, and i is odd. The case that i is even is left to the reader. Let us consider the last time unit t before the first task of w_i or of w_i^R will be executed. We distinguish between two possibilities according to the relative delay r of the executions of the prefixes up to t of J_1 to J_2 (i.e., the distance to the diagonal) in Graph($G_{\binom{i+1}{2}}$).

(1) Let the relative delay be at least i', i.e., the distance to the main diagonal is $r \geq i'$. If $r \geq i' + 1 = i$, we are done. If $r = i'$, then one can use the diagonal edges only to execute w_i or w_i^R, but because of the same parity of r and i', the induction hypothesis is satisfied. Since any change of the relative delay during the work on w_i or w_i^R causes a new delay, the hypothesis is true after processing w_i or w_i^R,

too.

(2) Let the relative delay r be at most i'. Then, following the induction hypothesis, the schedule contains in this moment at least i' delays if r is even, and at least $i'+1$ delays if r is odd. If r is even, then it is sufficient to observe that it is impossible to reach the border of the grid $G_{\binom{i+1}{2}}$ by using diagonal edges only. This is because $w_i = \binom{i}{2}+1, \ldots, \binom{i}{2}+i$, $w_i^R = \binom{i}{2}+i, \ldots, \binom{i}{2}+1$. Therefore, the execution of the task $\sigma_{\binom{i}{2}+j}$ is an obstacle for the following sequence of diagonal edges running parallel to the main diagonal in the distance $i-2j+1$ (corresponding to relative delay $i-2j+1$) for $j = 1, \ldots, \lfloor i/2 \rfloor$. Hence, at least 1 additional delay is necessary, and two additional delays are necessary if the schedule finishes in the same distance r from the diagonal.
If r is odd, and the schedule S executes w_i or w_i^R by using diagonal edges only, we have i "old" delays (induction hypothesis) and we are done. Obviously, if the distance to the diagonal changes, at least one additional delay occurs.

Consider an input instance $I_R^m = (\pi_1, \pi_2)$, for $m = k^2$, k a positive integer, where

$$\pi_1 = w_1, w_2, \ldots, w_k, u_{k-1}, u_{k-2}, \ldots, u_1,$$
$$\pi_2 = w_1^R, w_2^R, \ldots, w_k^R, u_{k-1}^R, u_{k-2}^R, \ldots, u_1^R,$$

where the w_i have the same meaning as before, and u_l is a sequence of l tasks for $l = 1, \ldots, k-1$, with u_l^R denoting the reverse of u_l . The example of I_R^9 for $\pi_1 = 1, 2, \ldots, m$ is given in Fig. 1. An extension of the analysis presented in Lemma 1 leads to the following result.

Lemma 2 *For every $m = k^2$, k a positive integer,*

$$makespan(I_R^m) \geq m + \sqrt{m} = m + k.$$

Proof: To prove the Lemma we show that every shortest path between the two opposite corners of the grid contains at least $2 \cdot k$ orthogonal grid edges; this implies that at least one of the two jobs is delayed by at least $k = \sqrt{m}$ time units and therefore, the makespan must be at least $m + \sqrt{m}$.

We use the induction of the proof of Lemma 1 in the following way. The prefixes $\pi_1' = w_1, w_2, \ldots, w_{k-1}$ and $\pi_2' = w_1^R, w_2^R, \ldots, w_{k-1}^R$ of the instance $I_R^{k^2}$ define an instance $I'^{\binom{k'+1}{2}}_R$ considered in Lemma 1, with $k' = k-1$. The suffixes $\pi_1'' = u_{k-1}, u_{k-2}, \ldots, u_1$ and $\pi_2'' = u_{k-1}^R, u_{k-2}^R, \ldots, u_1^R$ define the same instance in a symmetric way. We distinguish two cases.

(1) The relative delay r caused by the prefix $I'^{\binom{k'+1}{2}}_R$ is $r \leq k'$ and the parities of k' and r are the same. Then we know from Lemma 1 that any schedule of this prefix uses k' orthogonal grid edges. However, in the case that the parities of k' and r are the same it is impossible to reach the border of the grid $G_{\binom{k+1}{2}}$ by using diagonal edges only. This is because of w_k and w_k^R and hence, at least 1 additional delay is necessary, and two additional delays are necessary if the schedule finishes in the same distance r from the diagonal. After executing the tasks of w_k and w_k^R the schedule uses either $k'+1 = k$ orthogonal grid edges and changes the parity of r or it uses $k'+2 = k+1$ orthogonal grid edges and does not change the parity of r.

(2) The relative delay r caused by the prefix $I'^{\binom{k'+1}{2}}_R$ is $r \leq k'$ and the parities of k' and r differ. Then we know from Lemma 1 that any schedule of this prefix uses $k'+1$ orthogonal grid edges. In this case, the schedule can execute the tasks of w_k and w_k^R by using diagonal grid edges only and therefore, does not need to change the parity of r.

Now, if the parities of r and k' are the same and the schedule uses two additional delays to execute w_k and w_k^R then we have k' delays for the prefix and k' delays for the suffix, i.e., the sum-delay equals $2(k-1)+2 = 2k$. If the schedule uses only one additional delay to execute w_k and w_k^R then the parities of r and k' for the suffix differ. Hence, we have k' delays for the prefix and $k'+1$ delays for the suffix, i.e., the sum-delay equals $k+k-1+1 = 2k$. The case that the parities of r and k' differ for the prefix are symmetrical.

4. Upper bounds on the number of delays

In this section, we show that any input instance of *unit–J_m* can be scheduled with $2 \cdot m^{1-\varepsilon}$ delays for $d \leq m^{1/2-\varepsilon}$, as compared with the lower bound on the makespan. This improves on the upper bound $O(m+d)$ [5] for $d = o(m)$.

First, we give the upper bound for two jobs. Note that this upper bound meets the lower bound of Lemma 2.

Lemma 3 *For every positive integer m, any two job problem instance I of unit–J_m satisfies*

$$makespan(I) \leq m + \lceil \sqrt{m} \rceil.$$

Proof: For simplicity we present the proof for the case $m = k^2$ only. To do this we use the geometric representation. In what follows for $i = 0, 1, \ldots, \sqrt{m}$, the diagonal D_i of the grid G_m is the diagonal going from the position $(0, i)$ to the position $(m - i, m)$; similarly, diagonal D_{-i} goes from $(i, 0)$ to $(m, m-i)$, see Fig. 3.

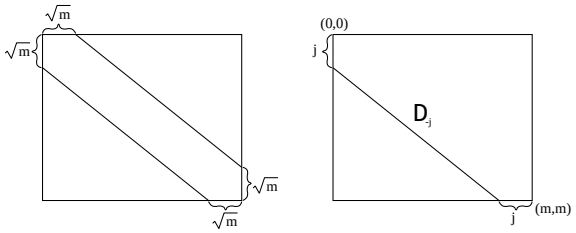

Fig. 3. The considered diagonals of G_m.

For each $i \in \{-\sqrt{m}, \ldots, 0, \ldots, \sqrt{m}\}$, we associate a schedule $S(D_i)$ to diagonal D_i. The schedule $S(D_i)$ uses first $|i|$ orthogonal grid edges to reach the beginning of the diagonal D_i, then it runs via this diagonal and avoids each obstacle on this diagonal by one horizontal move and one vertical move. Finally, it uses $|i|$ grid edges on the border of G_m in order to reach (m, m). Observe that the makespan of this schedule is exactly

$$m + |i| + \text{ the number of obstacles at } D_i$$

because the length of D_i is $m - |i|$ and the schedule uses $2 \cdot |i|$ steps to reach and to leave this diagonal. Therefore, the delay of the schedule $S(D_i)$ is $|i|+$ the number of obstacles at D_i. The sum of all delays over all $2\sqrt{m} + 1$ considered schedules D_i is at most

$$m + \sum_{i=-\sqrt{m}}^{\sqrt{m}} |i| = m + 2 \cdot \sum_{i=1}^{\sqrt{m}} i = m + \sqrt{m} \cdot (\sqrt{m} + 1)$$

because the number of all obstacles in the whole G_m is exactly m, the number of machines [1] . Since the average delay over all $2 \cdot \sqrt{m} + 1$ considered schedules is

$$\frac{m + \sqrt{m} \cdot (\sqrt{m} + 1)}{2 \cdot \sqrt{m} + 1} \leq \sqrt{m} + \frac{1}{2},$$

there must exist a schedule that has delay at most $\sqrt{m}$.

Now, we extend Lemma 3 to all input instances, i.e., any number of jobs.

Theorem 4 *For every positive integer m, and every instance I of unit–J_m with $d = o(m^{1/2})$ jobs, the length of any optimum schedule can be bounded from above by*

$$makespan(I) \leq m + 2d\sqrt{m} = m + o(m).$$

Proof: The idea of the proof is to generalize the case with $d = 2$ to any dimension. We can view the d-dimensional $m \times m \times \ldots \times m$ grid $G_{m,d}(I)$ as a subgrid of an infinite d-dimensional grid. We consider the following set $\mathcal{D}$ of diagonals that are parallel to the main diagonal of $G_{m,d}(I)$ that starts in the point $(0, 0, \ldots, 0)$ and ends in $(m, m, \ldots, m)$: We take every diagonal with a starting point $(i_1, i_2, \ldots, i_d)$, where there is exactly one $j \in \{1, \ldots, d\}$ such that $i_j = 0$, and $0 > i_b \geq -r$, for $b \in \{1, \ldots, d\} - \{j\}$ and some $m \geq r > 0$. Let $D(i_1, i_2, \ldots, i_d)$ denote the diagonal starting in the point with the coordinates $(i_1, i_2, \ldots, i_d)$ that ends in the point $(i_1 + m + a, i_2 + m + a, \ldots, i_d + m + a)$, where $a = \max\{|i_c| \mid c \in \{1, \ldots, d\}\} \leq r$. Every diagonal $D(i_1, i_2, \ldots, i_d)$ corresponds to a job schedule where the j^th job is postponed by i_j time units with respect to jobs starting with the delay 0. If this schedule reaches the final point $(i_1 + m + a, i_2 + m + a, \ldots, i_d + m + a)$ then all jobs were completely executed because $i_j + m + a \geq m$ for all $j \in \{1, \ldots, d\}$.

Obviously, the number of all such diagonals is exactly

$$d \cdot r^{d-1}. \tag{1}$$

Note, that one could consider also diagonals with starting points containing several 0 elements, but this makes the calculation more complex and the achieved gain is negligible.

Similarly, as in the 2-dimensional case we calculate an upper bound on the total delay of all $d \cdot r^{d-1}$ schedules. This bound can be obtained as the sum of an upper bound on the sum of the lengths of all diagonals and of an upper bound on the number of all delays occurring on these diagonals.

The starting points of all diagonals in $\mathcal{D}$ lie on the boundary of a grid that is mirrored on m diagonally (note, that the coordinates of such a starting point are all negative except for exactly one being equal to zero). At the end at most r extra diagonal steps are added to reach the end point at $(i_1 + m + a, i_2 + m + a, \ldots, i_d +$

[1] Therefore, in the worst case, all obstacles of G_m lie on the $2k + 1$ diagonals, see Fig. 3.

$m+a$). Therefore, the length of each described diagonal is bounded from above by $m + r$. Because of (1) and to make later calculations easier, the sum of the lengths of all diagonals is at most

$$d \cdot r^{d-1} \cdot (m + 2r). \tag{2}$$

Now, we count the number of possible delays. The d axes of the subgrid $G_{m,d}(I)$ are labeled by the d jobs. A label σ_i on an axis determines a $(d-1)$-dimensional subgrid of $G_{m,d}(I)$ of m^{d-1} d-dimensional unit grid cubes. An intersection of two such subgrids determined by the same label σ_i on two different axes is a $(d-2)$-dimensional subgrid of m^{d-2} d-dimensional unit grid cubes. Observe that the inner diagonal of any unit grid cube induced by this intersection subgrid as well as the corresponding diagonal on the surface of this unit grid cube are forbidden, for any schedule. Therefore, any of our diagonal schedules containing such a unit grid cube will get a delay. Obviously, if q $(d-1)$-dimensional subgrids labeled by σ_i meet in one unit grid cube, the diagonal schedule containing such a grid cube must use $q-1$ additional steps to avoid this obstacle.

We calculate the total number of delays as the sum of the number of delays caused by pairs of $(d-1)$-dimensional subgrids with the same label. We start with the following technical fact:

Fact 5 *The intersection of every pair of $(d-1)$-dimensional subgrids determined by the same task σ affects at most*

$$(d-1) \cdot r^{d-2}$$

diagonals of $\mathcal{D}$, each of them in exactly one unit grid cube.

Proof: It is obvious that every $(d-1)$-dimensional subgrid determined by a task σ intersects each of the diagonals of $\mathcal{D}$ in exactly one unit grid cube [1]. Thus, it remains to bound the number of diagonals intersecting the $(d-2)$-dimensional subgrid considered.

The intersection G_σ of two subgrids labeled by the same task σ corresponds to a fixed relative delay between the execution of two jobs. If the task σ is at the i^{th} position on the a^{th}-axis and at the j^{th} position on the b^{th} axis, $j \leq i$, then the relative delay between the execution of the b^{th} job and the a^{th} job is $j-i$ for all diagonals intersecting G_σ.

Thus, we count the number of diagonals from $\mathcal{D}$ with the relative delay $j-i$ between the b^{th} and the a^{th} job. Since $\mathcal{D}$ is the union of all $\mathcal{D}_p$'s, where $\mathcal{D}_p$ contains all diagonals with the p^{th} element equal to 0 and $\mathcal{D}_u \cap \mathcal{D}_v = \emptyset$ for $u \neq v$, $u, v \in \{1, 2, \ldots, d\}$, we count the number of such diagonals in $\mathcal{D}_p$ for every p separately.

Let $p \in \{1, 2, \ldots, d\} - \{a, b\}$. The intersection of $\mathcal{D}_p$ with G_σ meets all the diagonals with $D(c_1, c_2, \ldots, c_d)$, where $c_p = 0$ and $c_b = c_a + j - i$. One has r possible choices for every position from the $d-3$ positions of $\{1, 2, \ldots, d\} - \{p, a, b\}$, and at most $r - (j-i) \leq r$ choices for the a^{th} axis. The b^{th} axis is unambiguously determined by the a^{th} position. So, we have at most r^{d-2} grid cubes in the intersection of G_σ and $\mathcal{D}_p$ for $p \in \{1, 2, \ldots, d\} - \{a, b\}$. G_σ meets exactly the diagonals $D(t_1, t_2, \ldots, t_d)$ of $\mathcal{D}_b$, that has $t_b = 0$ and $t_a = i - j$. The number of such diagonals [2] is exactly r^{d-2}. G_σ does not intersect any diagonal from $\mathcal{D}_a$ because the diagonals $D(s_1, s_2, \ldots, s_d)$ in $\mathcal{D}_a$ have $s_a \geq s_u$ for every $u \in \{1, 2, \ldots, d\}$, i.e., the a^{th} job is executed as the first one and so it cannot be delayed with respect to any other job (including the b^{th} job). Thus, all together G_σ intersects at most

$$(d-1) \cdot r^{d-2}$$

diagonals.

Since we have m tasks in each of the d jobs and $\binom{d}{2}$ pairs of axes (jobs), the number of schedule delays on all $d \cdot r^{d-1}$ diagonals is at most

$$m \cdot \binom{d}{2} \cdot (d-1) \cdot r^{d-2}. \tag{3}$$

Therefore, the average number of delays per diagonal is at most

$$\frac{m \cdot \frac{d \cdot (d-1)}{2} \cdot (d-1) \cdot r^{d-2}}{d \cdot r^{d-1}} \leq \frac{m \cdot (d-1)^2}{2 \cdot r}.$$

Since the length of every diagonal is bounded by $m + 2r$, the average makespan over all diagonal strategies in $\mathcal{D}$ is bounded by

$$m + 2r + \frac{m(d-1)^2}{2r} \leq m + 2r + \frac{md^2}{2r}. \tag{4}$$

Choosing $r = d\sqrt{m}/2$ we obtain an average makespan over our dr^{d-1} diagonal strategies of at most

$$m + 2d\sqrt{m}.$$

[1] This is the cube that corresponds to the execution of the task σ in the job determined by the considered axis.

[2] with 2 fixed positions

Thus, there must exist at least one diagonal strategy with a makespan of at most $m + 2d\sqrt{m} = m + o(m)$ for $d = o(m^{1/2})$.

Corollary 6 *For every positive integer m and every instance I of unit–J_m with $d \leq m^{1/2-\varepsilon}$ jobs, with $0 < \varepsilon \leq 1/2$, the makespan of any optimal schedule can be bounded from above by*

$$makespan(I) \leq m + 2m^{1-\varepsilon}.$$

Proof: We choose

$$r = \lfloor \frac{1}{2} m^{1-\varepsilon} \rfloor,$$

and insert it into (4). Then we have

$$m + 2\lfloor \frac{1}{2} m^{1-\varepsilon} \rfloor + \frac{m(d-1)^2}{2\lfloor \frac{1}{2} m^{1-\varepsilon} \rfloor} \leq m + 2m^{1-\varepsilon}.$$

Since the best known upper bound on the makespan is $2(m+d) \geq 2m$, our upper bound is an improvement for $d = o(m)$.

5. A Randomized On-line Approximation Algorithm

In this section we consider the on-line version of our minimization problem which completely changes the scenario. In the classical scenario, one knows the whole input instance of an optimization problem and looks for a good solution. In the on-line scenario, one has to deal with the following tasks. One obtains only part of the input, and is forced to process this part. After one has solved it, one gets another part of the input that also has to be immediately processed. The input may be arbitrarily long. These kinds of tasks are called on-line problems, and the algorithms solving on-line problems are called on-line algorithms. The fundamental question posed in this framework is the following: How good can an on-line algorithm (that does not know the future) be in comparison to an algorithm that knows the whole input from the beginning? In our case, a good algorithm is an algorithm which computes a short schedule. Hence, one has a similar situation when dealing with optimization problems. In order to get a reasonable measure of the quality of on-line algorithms, we use the competitive ratio which essentially consists of comparing the costs $\mathrm{cost}_A(x)$ of solutions computed by an on-line algorithm A with the cost of the corresponding optimal solutions $\mathrm{Opt}(x)$ for every valid input x. The competitive ratio of an on-line algorithm for minimization problems is defined as

$$\mathrm{comp}_A(x) := \mathrm{cost}_A(x)/\mathrm{Opt}(x).$$

An algorithm A is called δ-competitive, if the competitive ratio is at most δ for every valid input x. In randomized algorithms, the competitive ratio may vary depending on the random decisions. Therefore, in this case we need the expected competitive ratio.

Let Z_x denote the random variable which measures the cost of the solution calculated by A on input x. We define the expected competitive ratio of A on x as

$$\mathrm{Exp–Comp}_A(x) = \frac{E[Z_x]}{\mathrm{Opt}(x)}.$$

A randomized algorithm A is $Exp[\delta]$-competitive, if $\mathrm{Exp–Comp}_A(x) \leq \delta$ for a real number δ and for every valid input x.

We propose a randomized on-line algorithm **Algorithm OLR_m**, that is given below, for *unit–J_m*. The number of jobs d and the number of machines m are known initially, with $d = o(m^{1/2})$. When the algorithm is executed, initially only the first task of each job is known. Every time the processing of a task is finished, the next task of the job is revealed. Thus at any time, the on-line algorithm only knows the next task of each job (and of course the completed tasks). Tasks occur in the order determined by their job and in arbitrary order across all jobs. The algorithm will choose uniformly at random a schedule, i.e. a particular distribution of initial delays among jobs.

Algorithm OLR_m

Input: The number of jobs d and the number of machines m are known initially and $d = o(m^{1/2})$. The tasks of the jobs are presented one by one, within each job in the order of their occurrence, and in arbitrary order across the jobs.

Step 1: Choose uniformly a diagonal D at random from $\mathcal{D}$, i.e., generate the start coordinates of a diagonal from $\mathcal{D}$ at random by following Theorem 1.

Step 2: Apply the schedule of the diagonal D randomly determined by Step 1 by avoiding the obstacles as they appear.

Theorem 7 *The randomized on-line algorithm OLR_m for unit–J_m*

(1) has an expected competitive ratio of at most $1 + 2d/\sqrt{m}$, that is, $1 + o(1)$ if $d = o(m^{1/2})$, and

(2) runs in linear time.

Proof: First we prove (ii). We have an input of length $m \cdot d$. A number $d \cdot \lceil \log_2 (d\sqrt{m}/2) \rceil$ of random bits is sufficient to determine a diagonal and therefore, Step 1 can be executed in linear time. It is straightforward to follow a given path for actual jobs (using diagonals whenever possible) in linear time.

Now, we prove (i). Since the average makespan over all schedules determined by the diagonals from $\mathcal{D}$ is at most $m + 2d\sqrt{m}$, and the optimum makespan is at least m, the expected approximation ratio of OLR is at most

$$\frac{m + 2d\sqrt{m}}{m} = 1 + \frac{2d}{\sqrt{m}}$$

Therefore, OLR is $(1 + 2d/\sqrt{m})$ competitive w.r.t. optimum schedules. Note that no (randomized) polynomial-time algorithm with an approximation ratio tending to 1 for $d = o(m^{1/2})$ with growing m has been known before. For $d \leq m^{1/2-\varepsilon}$ our algorithm is better than the 2-approximation algorithm of Leighton et al. [5]. Moreover, OLR_m shows nicely the power of randomization, because every deterministic on-line algorithm for *unit–*J_m has its competitive ratio at least $\Omega\big(d \,/\, \log_2 d\big)$ [5].

6. Randomization is more powerful than determinism in the on-line case

In this section we compare the expected competitive ratio of our randomized algorithm with the best competitive ratio achievable with deterministic algorithms. In order to analyze the competitive ratio of algorithms, it is common to treat an on-line problem as a game played by the algorithm designer against an adversary. The adversary knows the on-line algorithm and, if the algorithm is randomized, its probability distribution. Based on that, the adversary creates an input instance for the algorithm. Since in the deterministic case the knowledge about the on-line algorithm enables the adversary to determine the decisions of the deterministic on-line algorithms step by step, one can view the game between the algorithm designer and the adversary as follows. The adversary constructs an input instance for a given on-line algorithm. As on-line algorithms have to solve parts of the solution in order to get more input, the input instance is revealed piece by piece. Thus in the on-line version of *unit–*J_m, the adversary chooses the order of the tasks in the jobs.

For every deterministic on-line algorithm we design a successful adversary which causes the algorithm to calculate a schedule with a large delay. We also show that for two and three dimensions, it is not possible to substantially improve our results.

In a grid G_m, we define the **border** as the set of vertices $\{(i,j) : i = m \vee j = m\}$ in Graph(G_m). Further, **diag** denotes the number of diagonal steps out of the steps already taken by an algorithm while it is running. Analogously we will use **ort** for the number of orthogonal steps taken.

The strategy of the adversary is quite simple. He wants to create an instance such that at least every second step taken by the algorithm is not a diagonal one (see Fig. 4).

Adversary 1 (*Unit–*J_m**,** $d = 2$**)**

Input : *A deterministic on-line algorithm A for unit–J_m and the number of machines m.*

Step 1: *Place task 1 at the beginning of both jobs.*

Step 2: *While Algorithm A takes orthogonal steps and the border is not reached, place the available task with lowest order into the corresponding job, i.e., in the job in which a new task has to be revealed.*

Step 3: *If the last step of A corresponds to a diagonal step and the border is not reached then choose, for both jobs, the minimum task (according to its name) which is simultaneously available for both jobs (i.e. force one job to wait), and jump to Step 2.*

Step 4: *If a border is reached, fill up the job which is still not completely processed with the remaining tasks in an arbitrary order and end.*

Lemma 8 *For every deterministic on-line algorithm A, Adversary 1 constructs an input instance I_A such that the schedule computed by $A(I_A)$ contains at least $m/3$ delays.*

Proof: First we show that Adversary 1 creates a valid instance. Assume that there is a position (i,j) where the on-line algorithm advances diagonally (see Fig. 4) and at $(i+1, j+1)$ it is not possible to place an obstacle (i.e. to choose two identical tasks) and no border is reached. For all i', j', tasks starting at $(i+1, j')$ or $(i', j+1)$ can be excluded because of the monotonicity of advancing. The only remaining possibility is that there is no task left which is available on both axes (i.e. for both jobs). But this cannot be because if p is the next task in one job and q the next one in the other and $p > q$, then p is available for both jobs. This is a contradiction.

In order to bound the number of steps, we show that $diag/(diag + ort) \leq 1/2$ holds. The first step is forced to be orthogonal. The second is orthogonal or diagonal. Therefore after one and after two steps the inequality

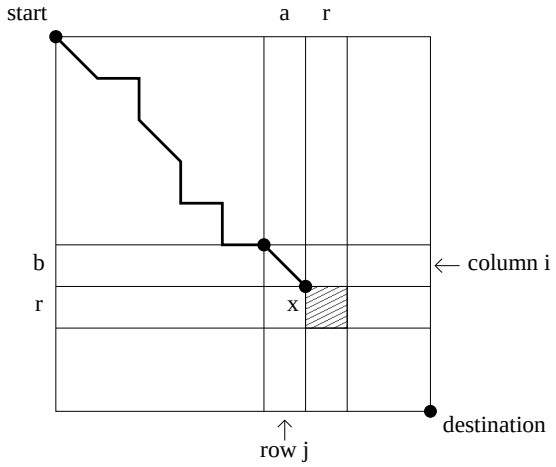

Fig. 4. After a diagonal step, Adversary 1 always forces an orthogonal step.

holds.

Let n the number of all diagonals before the actual position such that the inequality holds. Then the next step is blocked and an orthogonal step is forced. Let $l \geq 0$ be the number of steps until the next diagonal is used. For the actual position and the next diagonal we have $1/(1+l+1) \leq 1/2$. If no further diagonals follow, the ratio is 0.

When (m, m) is reached, the number of horizontal steps equals the number of vertical steps which implies $m = \mathit{diag} + \mathit{ort}/2$. Therefore we get $\mathit{diag} \leq 2/3m$ and $\mathit{ort} \geq 2/3m$. The number of steps increases when *diag* shrinks. Therefore at least $\frac{4}{3}m$ steps are necessary.

On the other hand the following deterministic on-line algorithm almost reaches this bound.

Algorithm Greedy-2d-*Unit-J_m*

Input: An instance of *unit–J_m* with $d = 3$.
Step 1: Whenever possible take a diagonal step.
Step 2: If an advance on both axes is possible,
  Step 2.1: If the actual position is not on the main diagonal, then take an orthogonal step towards the main diagonal.
  Step 2.2: Else take a horizontal step.
Step 3: Otherwise, when one of the jobs was completely processed, take the only possible step. (This is the case at the border.)

Lemma 9 *For every input instance I,*

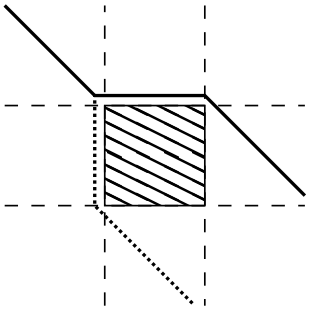

Fig. 5. Beside and under obstacles, diagonal steps are possible (unless the border is reached).

$$\mathrm{comp}_{\mathrm{Greedy-2d-}\mathit{Unit}-J_m}(I) \leq \tfrac{4}{3}.$$

Proof: We analyze the relative number of steps before a border is reached for an even number of steps. We will show $\mathit{diag}/(\mathit{diag} + \mathit{ort}) \geq 1/2$.

If at position $(0, 0)$ there is no obstacle, the algorithm starts with a diagonal step. An arbitrary second step suffices to reach the ratio. If at position $(0, 0)$ there is an obstacle, the second step is a diagonal one, because each row (and so the first row, too) may contain only one obstacle.

Let n be an even number such that after n steps the ratio holds. The next step either is taken diagonally or blocked (see Fig. 5). In the second case the second step is a diagonal one. In both cases after $n+2$ steps the ratio still holds. In order to complete the proof, we also have to show that the computed schedule is always close to the middle diagonal. Therefore we show that for each position (i, j) reached by the algorithm, $\mid i - j \mid \leq 1$ holds.

At position $(0, 0)$ this is obviously the case. Let the inequality hold for the first n steps and let (i', j') be the position reached after n steps. If the next step is a diagonal, the next position is $(i+1, j+1)$ and $\mid i+1-j-1 \mid = \mid i-j \mid$. Otherwise the next step is an orthogonal one. If $\mid i-1 \mid = 0$, after an orthogonal step the difference is one. If $\mid i-1 \mid = 1$ the next orthogonal step is taken in the direction of the main diagonal. After the step the difference is zero. At the border, only steps to the main diagonal are possible.

Because of the limited distance to the middle diagonal, the schedule can only reach the border at one of the positions in $\{(m-2, m-2), (m-1, m-2), (m-1, m-1), (m, m-1), (m, m)\}$. In all cases, only the

last step my be at the border. Apart from that, the ratio from above always holds and thus at most one additional step is needed.

On the other hand, if makespan$(I) = m$, then obviously also Greedy-2d-*Unit*-$J_m(I) = m$. In all other cases,

$$\mathrm{comp}_{\mathrm{Greedy-2d-}\mathit{Unit}-J_m}(I) \leq \frac{\frac{4m}{3}+1}{m+1} \leq \frac{4}{3}.$$

Now we analyze the case $d = 3$. Let A be a deterministic on-line algorithm for *unit–*J_m. We design the following adversary for A:

Adversary 2 (*Unit*-J_m, $d = 3$)
Let τ_x, τ_y and τ_z be the sets of all tasks that are not yet in the jobs x, y and z respectively. Placing a task into a job also implies to delete it from the set. Further let R be the set of jobs for which a new task has to be revealed. The set R is updated with each step taken by Algorithm A which is part of the input.

Input : *A deterministic on-line algorithm A for unit–J_m and the number of machines m.*
Step 1: *Place task 1 at the beginning of all three jobs.*
Step 2: *While $|R| \neq 3$ and $\tau_x \cup \tau_y \cup \tau_z \neq \emptyset$, for all $a \in R$ place the minimum task (according to its name) of τ_a in job a.*
Step 3: *If $|R| = 3$, place task $\min \bigcap_{a \in R} \tau_a$ in all jobs in R.*
Step 4: *If $\tau_x \cup \tau_y \cup \tau_z \neq \emptyset$, jump to 2.*

Informally, after an orthogonal step a 3 dimensional diagonal step can be accepted and directly after a 3 dimensional diagonal step, an obstacle follows that blocks all 3 dimensions and thus only orthogonal steps are possible.

On every axis, m steps have to be performed. Here we consider the sum of all steps taken on all axes. We call the steps performed on an axis of the grid **axis-steps** to distinguish them from the steps taken by the on-line algorithm. Thus every schedule has to perform $3 \cdot m$ axis-steps.

We call the different types of obstacles id-obstacle, where i is the number of jobs which have to process the same task. Thus an id-obstacle in the grid means that $d-i+1$ steps can be taken simultaneously. Analogously we use id-step for steps in i dimensions at once. The axes are numbered from 1 to d.

Lemma 10 *For every deterministic on-line algorithm A, Adversary 2 constructs an input instance I_A such that the schedule computed by $A(I_A)$ contains at least $m/2$ delays.*

Proof: Step 1 and Step 2 only require that the sets of available tasks are not empty. This property is given because otherwise the corresponding axis is not in R. It remains to show that it is always possible to place three tasks in Step 3. We do not have to consider the border because reaching the border implies $|R|$ to be smaller than 3. We show by induction over the number of 3d-steps that after a 3d-step, the most advanced job has been processed by all machines in $\{1, 2, \ldots, j\}$ for some $j \leq m$ and no other machine. Thus either task $j + 1$ is available for all jobs or the most advanced job has reached the border.

From the first step to the first 3d-step, the tasks are placed in increasing order. Thus the most advanced job has been processed exactly on the first j machines for some $j \leq m$ and if $j < m$, task $j + 1$ is available for all jobs.

Let i be the number of 3d-steps already taken, and let j' be the number of tasks already processed from the most advanced job directly after the i-th 3d-step. After the $(i + 1)$-th 3d-step, at least $j' + 1$ tasks of the most advanced job have been processed because a 3d-step advances on all 3 axes. All other tasks have been added in increasing order. Thus, if the new most advanced job is another one than before, all missing tasks of $\{1, 2, \ldots, j'\}$ have been added to the job before new tasks are placed. In any case, all new tasks are placed in increasing order. Therefore, after $i + 1$ 3d-steps, task $j' + 1$ is available for all jobs or the border is reached.

It only remains to show that the average number of axis-steps per step is at least 2. The first step of A is forced to be a 1d-step, and after all 3d-steps, a 1d-step follows or the border is reached. Let s_1, s_2 and s_3 be the number of performed 1d-steps, 2d-steps and 3d-steps respectively. Obviously, $s_3 \leq s_1$ holds. Then the average number of axis-steps per step is

$$\frac{s_1 + 2 \cdot s_2 + 3 \cdot s_3}{s_1 + s_2 + s_3} \leq \frac{s_1 + 2 \cdot s_2 + 3 \cdot s_1}{s_1 + s_2 + s_1} = 2.$$

Analogous to the previous case, we present the following deterministic on-line algorithm:

Algorithm Greedy-3d-*Unit*-J_m
Input: An instance of *unit–*J_m with $d = 3$.
Rule 1: If no obstacle is in the way, proceed on all three axes.
Rule 2: Avoid 2d-obstacles by proceeding simultaneously on the non-affected axis and on the less advanced axis of the remaining two, or the first of the

remaining axes, if both corresponding jobs have completed the same number of tasks.

Rule 3: Avoid 3d-obstacles by processing the least advanced jobs. If there are more than one least advanced jobs, proceed with first of them.

Lemma 11 *For every input instance I,*

$$\mathrm{comp}_{\mathrm{Greedy-3d-}\textit{Unit}\textit{-}J_m}(I) \leq \frac{3}{2} + \frac{6}{m}.$$

Proof: We will analyze the number of axis-steps performed in one time unit. Then we will show that the schedule keeps close to the main diagonal.

First we assume that no border is reached.

(a) After a 3d-obstacle (1d-step) a 2d-step is possible. Let i be the axis of this 1d-step. Then in the next step, on axis i a new machines has to process while on all axes $j \in \{1, \ldots, d\} - \{i\}$ still the old machine has to process.

(b) After a 2d-obstacle a 2d-step is possible, i.e., no 3d-obstacle is possible. Let i and j be the two axes with the same machine. Further let i be the axis on which A advances. In the next step, the machines corresponding to these axes have to process different tasks. Thus a 2d-step is possible again.

(c) Between two 3d-obstacles at least one 3d-step can be performed. This follows from (a) and (b) because the series of 2d-obstacles has to be interrupted.

Now we show that the distance between the most advanced job and the least advanced job is at most 2 steps. First we show that for the case where only 2d-obstacles are used. If 2d-steps are taken, each two consecutive 2d-obstacles cannot block the same two axes. Further there are always two possible steps when a 2d-obstacle is reached because one of the two blocked tasks can be chosen. A simple induction shows the result: At position (0,0,0), the distance to the main diagonal is zero. By distinguishing all possible sequences of 2d-obstacles (see Fig. 6, it follows that from any position (i, i, i) or $(i+1, i, i-1)$, within 3 steps either $(i+2, i+2, i+2)$ or $(i+3, i+2, i+1)$ is reachable. The distance between the schedule and the main diagonal never exceeds two steps.

If a 3d-obstacle is used, the direction of the first step is free selectable. Therefore every second 3d-obstacle gives the possibility to correct the distance to the main diagonal again. At the border, the most advanced job is finished. At most one additional step is caused by an odd number of 3d-obstacles. Altogether at most 3 steps are missing for each of the two remaining jobs to finish which means at most 6 additional steps. The properties

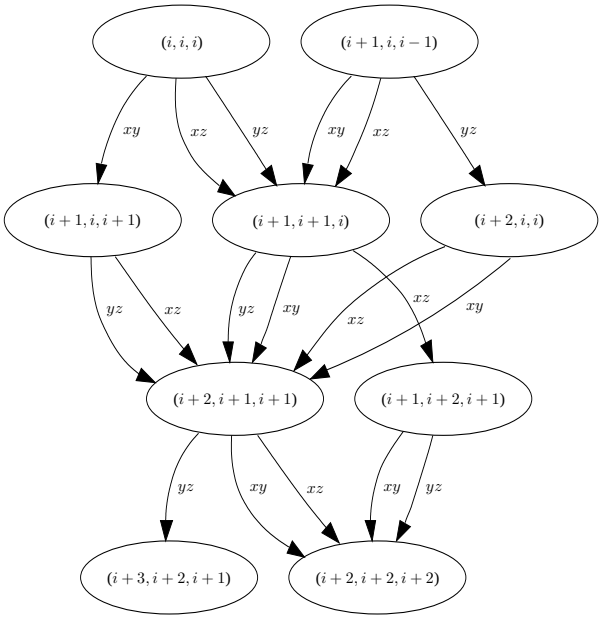

Fig. 6. From positions (i, i, i) and $(i + 1, i, i - 1)$, Alg. Greedy-3d-*Unit*-J_m reaches $(i + 2, i + 2, i + 2)$ or $(i + 3, i + 2, i + 1)$ within 3 steps, if an arbitrary sequence of 2d-obstacles has to be passed. The axes of the grid are denominated x, y and z. Note that two consecutive 2d-obstacle cannot be caused by the same pair of axes. The edges are labelled with those pairs of axes on which the schedule cannot proceed simultaneously.

(a), (b) and (c) are applicable for all remaining steps. Thus for each of these steps the average number of axis-steps performed is at least two. After $m + m/2$ time units, at least $3m - 6$ axis-steps are performed.

For $d > 3$ we can assign the first three axes as in the case $d = 3$. Thus the lower bound on the makespan of $m + m/2$ holds for every input instance with $d \geq 3$.

Theorem 12 *There is no deterministic on-line algorithm for unit–J_m with a competitive ratio better than $4/3$ for $d = 2$ or better than 1.5 for $d > 2$.*

Proof: As we have seen above, for an increasing m, OLR_m results in a makespan tending to m, i.e., the expected competitive ratio of this on-line algorithm tends to 1. This means that for every ε there is a m such that the competitive ratio of every deterministic on-line algorithm for *unit–J_m* is at least $4/3 - \varepsilon$ for $d = 2$ and $1.5 - \varepsilon$ for $d \geq 3$. This directly implies the theorem.

Thus randomization yields significantly better results for on-line *unit–J_m* than all deterministic approaches.

7. A deterministic approximation algorithm

As we already observed our grid representation provides an $O(m^d)$ algorithm for input instances with m machines and d jobs. The complexity of this algorithm is too large even for constant d's and it is not polynomial for d growing with m. The aim of this section is to present an efficient approximation algorithm at least for small d in comparison with m.

The idea is again to find a diagonal strategy, but in a deterministic way by looking on the $\binom{d}{2}$ 2-dimensional surfaces of $G_{m,d}(I)$ only. Remember that fixing a diagonal strategy is nothing else than fixing the relative delays between all pairs of jobs.

Algorithm SURFACE(I)

Input: $I = (J_1, J_2, \ldots, J_d)$, where J_i is the i^{th} job, i.e., a permutation of $(1, 2, \ldots, m)$, and $d \leq 1/2 \ \log_2 m$.

Step 1: If $d = 2$ take the best diagonal strategy from the $2\sqrt{m} + 1$ diagonal strategies with the relative delay between J_1 and J_2 bounded by $\sqrt{m}$. If $d > 2$, then apply SURFACE($J_1, J_2, \ldots, J_{d-1}$) in order to find a diagonal strategy D for $(J_1, J_2, \ldots, J_{d-1})$, that contains at most $2^{d-1}\sqrt{m}$ delays and for every $j \in 2, \ldots, d-1$ the relative delay between J_1 and J_j is at most $\sqrt{m}$. (Observe, that D fixes the delay between any two of the first $d-1$ jobs.)

Step 2: Fix consecutively the relative delays between J_d and the jobs $J_1, J_2, J_3, \ldots, J_{d-1}$ in the following way:

(2.1) Set S_1 as the set of the best[3] $\lfloor\sqrt{m}\rfloor$ diagonal strategies from the $2 \cdot \lfloor\sqrt{m}\rfloor + 1$ diagonal strategies for the input instance (J_1, J_d). (S_1 can be viewed as a set of relative delays from $\{\lfloor-\sqrt{m}\rfloor, \ldots, \lfloor\sqrt{m}\rfloor\}$ between J_1 and J_d and together with D it determines $\lfloor\sqrt{m}\rfloor$ diagonal strategies for $(J_1, J_2, \ldots, J_d)$).

(2.2) Set S_2 as the set of the best $\lfloor\sqrt{m}\rfloor/2$ diagonal strategies from the diagonal strategies of S_1 according to the input instance (J_2, J_d).

$\vdots$

(2.i) Set S_i as the set of the best $\lfloor\sqrt{m}\rfloor/2^{i-1}$ diagonal strategies from the diagonal strategies of S_{i-1} according to the input instance (J_i, J_d).

(2.d-1) Choose the best diagonal strategy $\overline{D}$ from S_{i-1} according to (J_{d-1}, J_d).

Output: The diagonal strategy determined by D and $\overline{D}$.

Theorem 13 *For every input instance $I = (J_1, J_2, \ldots, J_d)$ of unit–J_m with $d \leq 2\log_2 m$, the algorithm SURFACE(I)*

(i) *runs in time $O(d^2m^2)$, and*

(ii) *has an approximation ratio of at most $1 + \frac{2^d}{\sqrt{m}}$.*

Proof: SURFACE(I) does nothing else than looking on all $\binom{d}{2}$ 2-dimensional surfaces of $G_{m,d}(I)$ in order to choose a set of convenient delays with respect to every pair of jobs. The size of each surface is m^2 and the choice of a group of the best diagonals from a given set of diagonals can be done in $O(m^2)$ time. Thus, the overall time is in $O(d^2m^2)$.

To prove (ii) we first prove

(ii)$'$ The diagonal strategy computed by the algorithm SURFACE(I) contains at most $2^d\lceil\sqrt{m}\rceil$ delays.

We prove (ii)$'$ by induction on d. For $d = 2$ Lemma 3 guarantees at most $\lceil\sqrt{m}\rceil$ delays. Let (ii)$'$ be true for $d-1$, i.e., the strategy D computed for $(J_1, J_2, \ldots, J_{d-1})$ in the first step of SURFACE(I) contains at most $2^{d-1} \cdot \lceil\sqrt{m}\rceil$ delays between the first $d-1$ jobs. In Step (2.1) we look on the surface determined by (J_1, J_d). Following Lemma 3 the average number per diagonal of obstacles on the main $2 \cdot \lfloor\sqrt{m}\rfloor + 1$ diagonals of this surface is at most

$$\frac{m}{2 \cdot \lfloor\sqrt{m}\rfloor + 1} \leq \frac{\lceil\sqrt{m}\rceil}{2}.$$

So, there must exist a set S_1 of $\lfloor\sqrt{m}\rfloor$ diagonals such that every diagonal of S_1 has at most $\lceil\sqrt{m}\rceil$ obstacles, i.e., at most twice the average. Observe, that each of these diagonals from S together with D determines a diagonal strategy for the whole instance $I = (J_1, J_2, \ldots, J_d)$, where J_1 and J_d have at most $\lceil\sqrt{m}\rceil$ delays. Thus, we have $\mid S_1 \mid = \lfloor\sqrt{m}\rfloor$ candidates for the output. In Step (2.2) we choose the best $\lfloor\sqrt{m}\rfloor/2$ from these candidates with respect to the obstacles for J_2 and J_d. Since these $\lfloor\sqrt{m}\rfloor$ candidates can contain together at most m obstacles, the average number of obstacles is $\lceil\sqrt{m}\rceil$, and so there exist $\lfloor\sqrt{m}\rfloor/2$ diagonals each with at most $2 \cdot \lceil\sqrt{m}\rceil$ obstacles. In general, in Step (2.i) for $2 \leq i \leq d-2$ we choose from the remaining $\lfloor\sqrt{m}\rfloor/2^{i-2}$ candidates the best $\lfloor\sqrt{m}\rfloor/2^{i-1}$ candidates with respect to

[3] with respect to the number of obstacles

the number of obstacles on the surface determined by J_i and J_d. Each of the candidates of S_i has at most $2^{i-1} \cdot \lceil \sqrt{m} \rceil$ obstacles between J_i and J_d. The last Step (2.d-1) corresponds to the choice of the best diagonal D' (with respect to the relation between J_{d-1} and J_d) from $\lfloor \sqrt{m} \rfloor / 2^{d-3}$ candidates. The number of obstacles between J_{d-1} and J_d on D' is bounded by the average

$$\frac{m}{\lfloor \sqrt{m} \rfloor / 2^{d-3}} = 2^{d-3} \cdot \lceil \sqrt{m} \rceil.$$

Let $\overline{D}$ be the resulting strategy for I. Thus, the overall number of obstacles between J_d and all other jobs in $\overline{D}$ is at most

$$\sum_{i=1}^{d-2} 2^{i-1} \cdot \lceil \sqrt{m} \rceil = (2^{d-2} - 1) \cdot \lceil \sqrt{m} \rceil < (2^{d-1} - 2) \cdot \lceil \sqrt{m} \rceil.$$

By the induction hypothesis the number of obstacles between the first $d-1$ jobs is at most $2^{d-1} \cdot \lceil \sqrt{m} \rceil$, and therefore, the overall number of obstacles on all $\binom{d}{2}$ 2-dimensional surfaces is at most

$$(2^d - 2) \cdot \lceil \sqrt{m} \rceil.$$

Obviously, these obstacles together cause at most $(2^d - 2)\lceil \sqrt{m} \rceil$ delays when following the diagonal strategy $\overline{D}$. The length of $\overline{D}$ is at most $m + 2 \cdot \lfloor \sqrt{m} \rfloor$ because $\overline{D}$ was constructed in such a way that no relative delay between J_1 and any other job would be greater than $\lfloor \sqrt{m} \rfloor$ (i.e., the relative delay between any pair of jobs is at most $2 \cdot \lfloor \sqrt{m} \rfloor$). Thus, the schedule that follows $\overline{D}$ has a makespan of at most

$$m + 2 \cdot \lfloor \sqrt{m} \rfloor + (2^d - 2) \cdot \lceil \sqrt{m} \rceil \leq m + 2^d \cdot \lceil \sqrt{m} \rceil.$$

Since the optimum makespan is at least m, the approximation ratio is at most

$$1 + \frac{2^d}{\lfloor \sqrt{m} \rfloor}.$$

The main point is that SURFACE works in quadratic time for constant d and can provide a good approximation ratio in that case. Observe, that the approximation ratio of SURFACE(I) tends to 1 with growing m for $d = o(\log_2 m)$.

8. Conclusions

For the job shop schedule problem *unit–J_m* we derived an upper bound on the makespan of optimum schedules that improves on the result given in [5] for $d = o(m^{1/2})$. We presented a competitive w.r.t. the makespan of an optimum solution, randomized on-line approximation algorithm that solves *unit–J_m* in linear time with an expected approximation ratio of $1 + 2d/\sqrt{m}$ which amounts to $1+o(1)$ for $d = o(m^{1/2})$. For $d = o(m^{1/2})$ the algorithm is the best approximation algorithm for *unit–J_m*. We showed that every deterministic on-line algorithm for *unit–J_m* yields significantly worse results than our randomized algorithm. If m is arbitrarily large, the competitive ratio w.r.t the makespan is at least 1.5 for every deterministic on-line algorithm, if $d \geq 3$. Our deterministic approximation algorithm is efficient at least for small d's in comparison with m. Its run-time is $O(d^2 m^2)$, and it has an approximation ratio of at most $1 + \frac{2^d}{\lfloor \sqrt{m} \rfloor}$ which tends to 1 with growing m for $d = o(\log_2 m)$. For the special case of *unit–J_m* with two jobs, which is solvable in linear time, we have shown that there exist input instances such that every schedule has a makespan of at least $m + \sqrt{m}$. Therewith, we proved that our upper bound on the makespan for $m + \lceil \sqrt{m} \rceil$, for $d = 2$ cannot be improved.

Received 8 February 2006; revised 25 July 2006; accepted 28 August 2006

References

[1] Brucker, P.: An Efficient Algorithm for the Job Shop Problem with Two Jobs. *Computing*, 40:353–359, 1988.

[2] Feige U., Scheideler, C.: Improved Bounds for Acyclic Job Shop Scheduling. *Proc. 28th ACM Symposium on Theory of Computing*, pp. 624–233, 1998.

[3] Goldberg, L.A., Paterson, M., Srinivasan, A., Sweedyk, E.: Better Approximation Guarantees for Job-shop Scheduling. *Proc. 8th ACM-SIAM Symposium on Discrete Algorithms*, pp. 599–608, 1997.

[4] Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet Routing and Job-Shop Scheduling in $O(Congestion + Dilation)$ steps. *Combinatorica*, 14:167–186, 1994.

[5] Leighton, F.T., Maggs, B.M., Richa, A.W.: Fast Algorithms for Finding $O(Congestion + Dilation)$ Packet Routing Schedules. *Combinatorica*, 19:375–401, 1999.

[6] Lenstra, J.K., Rinnooy Kan A.H.G.: Computational Complexity of Discrete Optimization Problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.

[7] Scheideler, C.: *Universal Routing Strategies for Interconnection Networks*. Lecture Notes in Computer Science 1390, Springer Verlag, 1998.

[8] Shmoys, D.B., Stein, C., Wein, J.: Improved Approximation Algorithms for Shop Scheduling Problems. *SIAM J. on Computing*, 23:617–632, 1994.

[9] Williamson, D.P., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevast'janov, S.V., Shmoys, D.B.: Short Shop Schedules. *Operations Research*, 45:288–294, 1997.