Scheduling with safety distances*

Frits C.R. Spieksma
Department of Mathematics, University of Limburg, P.O. Box 616, 6200 MD Maastricht, The Netherlands

Gerhard J. Woeginger
Institut für Theoretische Informatik, TU Graz, Klosterwiesgasse 31/II, A-8010 Graz, Austria

and

Zhongliang Yu☆
Institut für Mathematik B, TU Graz, Kopernikaugasse 24, A-8010 Graz, Austria

We investigate the problem of Scheduling with Safety Distances (SSD) that consists in scheduling jobs on two parallel machines without machine idle time. Every job is already assigned to its machine, and we just have to specify an ordering of the jobs for each machine. The goal is to find orderings of the jobs such that the minimum time elapsed between any two job completion times is maximized. We prove that this problem is NP-hard in general and give polynomial time algorithms for special cases. These results combined establish a sharp borderline between NP-complete and polynomial solvable versions of the problem SSD.

1. Introduction

In this paper, we investigate the following problem, called *Scheduling with Safety Distances* or SSD for short. We are given two identical parallel machines M_1 and M_2. Associated with every machine M_j is a non-empty set J_j of jobs that have to be processed on M_j, $j = 1, 2$. All job processing times are positive integers and, except when explicitly mentioned otherwise, we will identify a job with its processing time. The goal is to find a schedule S for all jobs such that

$$\min\{|C^S(j_1) - C^S(j_2)| : j_1, j_2 \text{ are jobs}\} \geq d,$$

*This research was supported by the Christian Doppler Laboratorium für Diskrete Optimierung.
☆On leave from the Mathematics Section, Forestry University Nanjing, Nanjing, PR China.

where d is some positive integer and $C^S(p)$ denotes the completion time of job p in schedule S. No machine idle time is allowed and it is assumed that both machines start processing the jobs at time 0. The number d will be called the *safety distance*.

The problem SSD is a real-world problem and arises in the following context [5]. In some chemical production process, there are three machines A, B and C and lots of jobs of two types. A job of type one first spends time a_i on machine A and *immediately afterwards* (no wait) it must spend time c on machine C, where time a_i depends on the job but c does not. Similarly, jobs of the second type spend time b_i on machine B and immediately afterwards they are processed on machine C for c time units. Machines A and B work at high temperatures and should be in use without interruption. Machine C performs a kind of package work finishing the job. This packing must be done immediately as a job leaves machine A, respectively, B. On machine C, idle time is allowed.

If we translate the above restrictions into a more mathematical formulation, we see that the processing on machine C forces job completion times on A and B to be at least c time units away from each other. We arrive at problem SSD with $M_1 = A$, $M_2 = B$, J_1 (J_2) is the set of jobs of type one (two) and $d = c$.

Another environment in which this problem arises is the area of tool handling in flexible manufacturing cells. A simplified version of a problem described in Agnetis et al. [1] is as follows. Consider a machine-cell with two machines and a single robot serving both machines. The task of the robot consists of supplying each machine with the appropriate tool needed to process some part on a machine, and, after completion of a part, removing this tool. These tools are stored in a central tool magazine, whereas each individual machine has no possibility to store one or more tools. Therefore, a tool has to be delivered to and removed from a machine exactly at the moment when the machine starts processing a new part. It takes the robot e time units to transport a tool from the magazine to the machine or vice versa. If no idle time is allowed on the machine, it is not difficult to see that this problem is equivalent to SSD with $d = 2e$.

1.1. OUR RESULTS

Although the restrictions seem to be rather strong at first sight and SSD appears to be a "simple" problem, we will prove that deciding the existence of a schedule with safety distance d is NP-complete even if

- the safety distance d equals 1 (and is not part of the input!)
  or
- the safety distance is part of the input, but all jobs in J_1 have equal length.

Moreover, we will identify some special cases for which the problem SSD is solvable in polynomial time. For example, we develop a polynomial time algorithm for the

case where $d = 1$ and where all jobs in J_1 are of equal length (i.e. where both of the two above conditions hold at the same time). By our results, a sharp borderline between NP-completeness and polynomial solvability of the problem SSD is established.

1.2. RELATED RESULTS

The results we are aware of that are related to our problem SSD concern two problems, namely scheduling of periodical events on a cycle (see Guldan [7], Burkard [3], Brucker et al. [4], and Brucker and Meyer [2]) and scheduling problems with a single server (see Hall et al. [8]). In the former problem, one is given a set of polygons with all vertices on a common cycle. The vertices correspond to jobs, and the goal is to rotate the polygons in such a way that the minimum distance between neighboring jobs on the cycle is maximized. This minimum distance is called the *minimal safety interval* (and also gave the name to SSD). The problem arises in connection with scheduling subways using the same routes. Trains run in different intervals, but for safety reasons there should be a certain time interval between two consecutive trains. Burkard [3] deals with the problem of scheduling two regular polygons on a cycle and gives an exact solution based on number-theoretic arguments.

The problem considered in Hall et al. [8] involves m parallel machines, n jobs and a single server which is needed to load a job on a machine. Simultaneous requests by machines for the server may result in machine idle time. They study the complexity of this problem for different objectives, different job characteristics and different possibilities for the setup time. SSD differs from their setting in the sense that, for SSD, an assignment of jobs to the machines is prespecified, and machine idle time is not allowed.

1.3. ORGANIZATION OF THE PAPER

In section 2 we give our NP-completeness proofs, and in section 3 some polynomial time results for special cases are stated. Section 4 finishes with the discussion.

2. NP-completeness results

In this section, we prove two NP-completeness results. This is done by two similar reductions from the Three Partitioning Problem (see Garey and Johnson [6]) which is known to be NP-complete in the strong sense and which is defined as follows.

Instance: A positive integer U and a set $A = \{a_1,\ldots,a_{3n}\}$ of $3n$ integers with $U/4 < a_j < U/2$, for $1 \leq j \leq 3n$.

Question: Is it possible to partition A into n triples such that the numbers in each triple exactly sum up to U?

THEOREM 2.1

Let J_1 and J_2 be two job sets to be processed on machines M_1 and M_2, respectively, and let d be a given safety distance. Then it is NP-complete to decide whether there exists a feasible schedule without machine idle time and with safety distance d even if

(i) all jobs in J_1 have equal length, or if

(ii) $d \geq 1$ is fixed and not part of the input.

Proof

We start with the proof of (i). Consider some instance of Three Partitioning. Without loss of generality, we may assume that all numbers a_i $(1 \leq i \leq n)$ and U are divisible by 16 (and otherwise, we simply multiply all of them by 16). We construct two job sets J_1 and J_2 as follows:

- J_1 contains $n + 1$ jobs of length $3U/2$, and
- J_2 contains $n + 1$ jobs of length $U/2$, two jobs of length $U/2 - 1$, one job of length $U/4 + 2$, and $3n$ jobs with lengths a_j, $1 \leq j \leq 3n$.

We claim that there exists a schedule with safety distance $d = U/4$ if and only if the partitioning problem has a solution.

(if) Assume that Three Partitioning has a solution. Consider the following schedule for M_2: first, we process the jobs of length $U/2 - 1$, $U/2 - 1$, $U/4 + 2$ and $U/2$ and then we sequence alternatively a triple of jobs whose processing times occur in a triple of the solution to Three Partitioning, and a job of length $U/2$. It is easy to check (see figure 1) that this sequence has safety distance $U/4$.

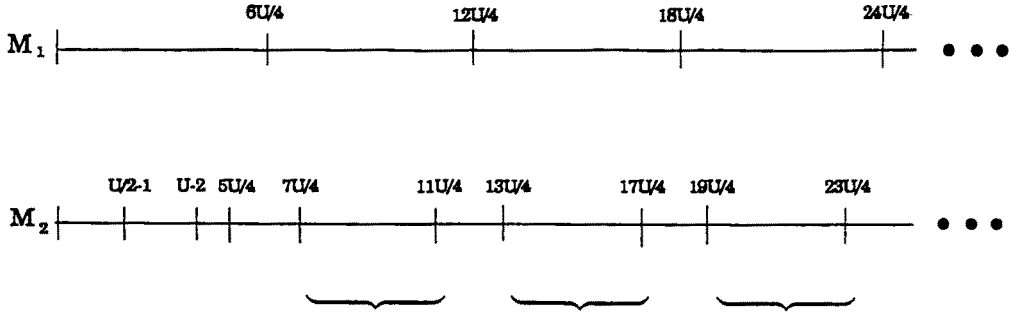

Figure 1. A schedule with safety distance $U/4$.

(only if) Suppose a schedule with safety distance $d = U/4$ exists. Since the processing times of the jobs of J_1 are equal, the completion times of these jobs must be the numbers $\langle 3U/2, 3U, \ldots, 3(n+1)U/2\rangle$. It follows that there are $n+1$ regular returning intervals of length $U/2$ (starting $U/4$ time units before $3iU/2$ and ending $U/4$ time units after time $3iU/2$, for $1 \le i \le n+1$) in which no job from the set J_2 is allowed to start. Since all other jobs in J_2 are of length strictly smaller than $U/2$, it is easy to see that the only way to get a feasible schedule for the jobs in J_2 is to start the $n+1$ jobs of length $U/2$ at times $(\frac{3}{2}i - \frac{1}{4})U$, $i = 1,\ldots,n+1$. This implies that for machine M_2 there remains an interval of length $5U/4$ at the beginning and n intervals of length U between any two consecutive jobs of length $U/2$ on machine M_2.

The only remaining jobs in J_2 whose lengths are not divisible by 4, are the two jobs with length $U/2 - 1$ and the single job of length $U/4 + 2$. All $n+1$ interval lengths are divisible by 4; it is not hard to check that for this reason the three jobs must be scheduled into a common interval. However, they have overall length $5U/4$ and thus they will only fit into the first interval. This leaves n intervals, each of length U, which must be filled by the jobs with lengths a_j, $1 \le j \le 3n$. Hence, if a schedule with safety distance $d = U/4$ exists, we have also found a solution to the instance of Three Partitioning. This completes the proof of (i).

Next, we will show that SSD is NP-complete for safety distance $d = 1$. Since all job processing times are integers, demanding safety distance $d = 1$ is equivalent to demanding that all job completion times are pairwise distinct.

Given an instance of Three Partitioning as above, we define another instance of SSD where

- J_1 contains n jobs of length $U+2$ and $(n+1)U+1$ jobs of length 1, and
- J_2 contains n jobs of length $U+2$ and $3n$ jobs with lengths a_j, $1 \le j \le 3n$.

Again, we claim that there exists a feasible schedule (with safety distance $d = 1$) if and only if the partitioning problem has a solution.

(if) On machine M_1, we first schedule $U+1$ jobs of length 1 and then we sequence alternatively a job of length $U+2$ and U jobs of length 1. On machine M_2, we sequence alternatively a job of length $U+2$ and a triple of jobs whose processing times occur in a triple of the solution to Three Partitioning. It is easy to verify (see figure 2) that no two completion times coincide in this schedule.

(only if) Suppose SSD has a feasible solution. Consider the jobs of length $U+2$ of J_1 in the feasible schedule. They induce $n+1$ intervals in which the jobs of length 1 are positioned. Into the first of these intervals (starting at time zero and ending as the first length $U+2$ job is processed), one can place at most $U+1$ jobs of length 1; otherwise, no feasible schedule on machine M_2 can be constructed, since all jobs in J_2 have length at most $U+2$. The same argument implies that in all other intervals, no more than U jobs of length 1 can be placed. Since there are $(n+1)U+1$

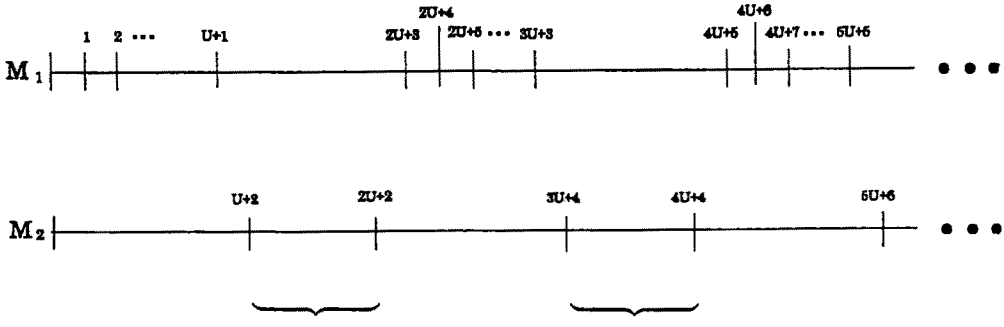

Figure 2. A schedule with safety distance 1.

jobs of length 1, it follows that each interval must contain its maximum number of jobs of length 1.

Moreover, these intervals on M_1 determine the positions of the jobs of length $U+2$ on M_2: these are the only jobs sufficiently large to bridge the intervals; since no idle time is allowed, they must bridge the first n intervals. Then on M_2, there remain n gaps each of length U which have to be filled with the jobs of length a_j, $1 \leq j \leq n$. Thus, if SSD has a feasible solution, then Three Partitioning must have one, too.

This completes the proof of claim (ii) for $d = 1$. In case the safety distance is any other fixed integer $d \geq 2$, we perform the above construction with all processing times multiplied by d. □

Remark 2.2

The job set J_1 in the proof of theorem 2.1(ii) contains only jobs with *two* distinct processing times. In the next section, we will show that essentially this is the strongest possible result, since the case of a set J_1 with identical processing times and $d = 1$ is solvable in polynomial time.

3. Polynomially solvable special cases

In this section, we present two results on special cases of SSD with safety distance $d = 1$ that are solvable in polynomial time: (1) First, if all jobs in $J_1 \cup J_2$ have pairwise distinct processing times, then there exists a simple necessary and sufficient condition for the existence of a schedule with safety distance $d = 1$. (2) Secondly, the case is treated where all jobs in J_1 have equal length, while the job lengths in J_2 are arbitrary.

3.1. PAIRWISE DISTINCT PROCESSING TIMES

We denote by P_j the total processing time of all jobs in J_j, $j = 1, 2$ (and identify each job p_i with its length).

LEMMA 3.1

Assume that all jobs $p_i \neq p_j$ in $J_1 \cup J_2$ fulfill $|p_i - p_j| \geq 2d - 1$ and $|p_i| \geq d$. Then there exists a schedule with safety distance d if and only if $|P_1 - P_2| \geq d$ holds. Moreover, this schedule can be constructed in linear time.

Proof

The (only if) part is trivial, since otherwise in every schedule the completion times of the final job on M_1 and of the final job on M_2 would violate the safety distance.

For the (if) part, suppose that there are two job sets satisfying $|P_1 - P_2| \geq d$, which cannot be scheduled at safety distance d, and consider the counterexample with the smallest total number of jobs. Without loss of generality, we assume $P_1 > P_2$.

(Case 1) Suppose $|J_1| \geq 2$. If we delete an arbitrary job p from J_1, either (1.i) the inequality $|P_1 - p - P_2| \geq d$ holds and a feasible schedule exists for the new instance (since we still have $|p_i - p_j| \geq 2d - 1$ for all i, j, and since our counterexample was minimal), or (1.ii) $|P_1 - p - P_2| < d$ holds. In case (1.i), we add p at the end of the feasible schedule for the smaller instance, we find a feasible schedule for J_1 and J_2 and arrive at a contradiction. In case (1.ii), we remove another job $q \neq p$ from J_1. Similarly as above $|P_1 - q - P_2| < d$ must hold. Combining this with $|P_1 - p - P_2| < d$ yields $|p - q| \leq 2d - 2$, a contradiction to our assumptions.

(Case 2) Now suppose that in the counterexample $|J_1| = 1$ and let $J_1 = \{p\}$. Since $p = P_1 \geq P_2 + d$, it is easy to check that *every* schedule is a feasible schedule with safety distance d.

It remains to show that a feasible schedule can be computed in time linear in the number n of jobs. A simple algorithm is implicit in the above combinatorial discussion: In a preprocessing step, the jobs in J_1 and J_2 are sequenced arbitrarily and the total processing times P_1 and P_2 are computed. Clearly, this can be performed in $O(n)$ time. The stop criterion is fulfilled if $|J_1| = 1$ or $|J_2| = 1$. In this case, the precomputed sequence constitute an optimum schedule (since any sequence constitutes an optimum schedule). As long as the stop criterion is not fulfilled, consider the job set J_j with $P_j > P_{3-j}$ and let p and q be the two first jobs in the precomputed sequence for J_j. By the above considerations, one of these two jobs – w.l.o.g. job p – fulfills $|P_j - p - P_{3-j}| > d$. This job p is scheduled last on machine M_j, it is removed from the sequences and P_j is updated. This step is performed in constant time per job.

Summarizing, the preprocessing takes $O(n)$ time and then the jobs are scheduled one by one in constant time per job until $|J_1| = 1$ or $|J_2| = 1$ holds. □

Remark 3.2

The statement in lemma 3.1 is best possible in the following sense. The job sets $J_1 = \{3d, 5d\}$ and $J_2 = \{7d + 1, 9d - 1\}$ fulfill (i) $|p_i - p_j| \geq 2d - 2$ for all $i \neq j$, (ii) $|p_i| \geq d$ for all i, and (iii) $|P_1 - P_2| \geq d$. Nevertheless, no feasible schedule with safety distance d exists.

COROLLARY 3.3

In case all job processing times in $J_1 \cup J_2$ are pairwise distinct, a schedule with safety distance $d = 1$ exists iff $P_1 \neq P_2$. □

3.2. ALL JOBS ON M_1 HAVE EQUAL LENGTH

In this subsection, we consider the special case where the safety distance $d = 1$, and where all jobs of J_1 have equal length, say k. Since in our arguments we will have to distinguish between a job p and its processing time $length(p)$, from now on we drop the convention of identifying a job with its processing time.

First, we will restrict ourselves to instances of this special case satisfying the following three conditions.

(C1) $P_1 > P_2$ (this ensures that all completion times of the form ik are forbidden for jobs on M_2).

(C2) $t \cdot k < P_2 < (t + 1) \cdot k$ for some $t \in \mathbb{N}$ (since it is obvious that otherwise no feasible sequence for the jobs in J_2 can exist).

(C3) $1 \leq length(p) \leq k - 1$ for all jobs p in J_2.

Let n denote the number of jobs in J_2, let S refer to a sequence of the n jobs in J_2, and let the latest starting time of a job in S before or on time ik, $i = 0,\ldots,t + 1$ be denoted by q_i (notice that $q_0 = 0$). If, for some sequence S, $q_i = ik$ for some $i \geq 1$, we say that S has a *hit* at $i \cdot k$. A sequence without hits is called a *hitfree* sequence (and constitutes a feasible solution to the problem).

Consider the algorithm depicted in figure 3 which takes as input an arbitrary sequence S of the jobs in J_2 with their processing times and the resulting q_i values for $i = 0,\ldots,t + 1$. An informal description of the algorithm is as follows. Given some sequence S of the jobs in J_2, the largest hit is identified. Consider the job starting at that hit, and say that it has length x. Now, by exchanging this job with any job before the last hit which has length not equal to x, we avoid this hit and do not induce later hits (step 1). This step is repeated until a hitfree sequence is found or a sequence is constructed in which all jobs before the latest hit and the job starting at that hit have the same length x. In the latter case, the algorithm finds the first job in the current sequence whose length is not equal to x and places it directly in front of the job currently ending at the *first* hit, thereby avoiding this first hit (step 3). Again,

Step 0: $i := t$; *stopstep*1 :=false; go to Step 1.

Step 1: while $q_i < i \cdot k$ do $i := i - 1$;
if $i = 0$ then stop, S has safety distance 1, else
begin
let s be the job in J_2 starting at $i \cdot k$;
$x := lenght(s)$;
if there is a job in $[0, i \cdot k]$, say job r, with $length(r) \neq x$, then
begin
interchange job r and job s;
update the q_i values for all i;
end else *stopstep*1 :=true;
end;
if not *stopstep*1 then go to Step 1, else go to Step 2.

Step 2: $j := 1$; go to Step 3.

Step 3: while $q_j < j \cdot k$ do $j := j + 1$;
find the first job in S starting after q_j, say job z, with $length(z) \neq x$;
if no such job exists, stop: no sequence of the jobs in J_2 exists which has safety distance 1, else
begin
place, in S, job z directly in front of the job currently ending at q_j;
update the q_j values for all j;
end;
if S has safety distance 1, stop, else go to Step 3.

Figure 3. An algorithm for finding a hitfree sequence.

this step is repeated until a hitfree sequence is found or no job of length not x exists after the current first hit.

In order to prove that the algorithm indeed finds a hitfree sequence (if one exists), we need to introduce some additional definitions.

- We denote the number of different job lengths present in J_2 by D, and by $p_1,\ldots,p_D$ these different job lengths.
- We define for $i = 1,\ldots,D$, n_{p_i} as the number of jobs in J_2 with length p_i.
- We define for $i = 1,\ldots,D$ and for all jobs j in J_2, $l_j^{p_i}$ as the maximal number of jobs of length p_i which, when sequences consecutively following job j which starts at time unit $-p_i$, do not induce a hit.
- We define for $i = 1,\ldots,D$, $l_0^{p_i}$ as the maximal number of jobs of length p_i which, when starting at time 0, do not induce a hit.
- Finally, we define for $i = 1,\ldots,D$, $M_{p_i} = l_0^{p_i} + \sum_{j \in J_2} l_j^{p_i}$.

Notice that $l_j^{p_i} = 0$ if $length(j) = p_i$. Also, $l_j^{p_i}$ may go to infinity for some p_i and $length(j)$. In fact, it is not difficult to see that, for each i, j, $l_j^{p_i}$ can be computed by

solving the equation $\gamma p_i - \delta k = p_i - p_j$ for some $\gamma, \delta \in \mathbb{N}$. This can be done by the Euclidean algorithm (see Nemhauser and Wolsey [10]). If the equation has a solution, we set $l_j^{p_i}$ equal to the smallest nonnegative value of γ for which a solution exists. Further, it follows from these definitions that if some job j starts at $-Kp_i$ for some $K \in \mathbb{N}$ (or at $qk - Kp_i$ for some q, $K \in \mathbb{N}$), then the maximal number of jobs of length p_i which can follow this job j without producing a hit equals $l_j^{p_i} + K - 1$.

LEMMA 3.4

Under conditions (C1) through (C3), a hitfree sequence exists if and only if $n_{p_i} \leq M_{p_i}$ for all i.

Proof

(only if) Suppose that for some i, $1 \leq i \leq D$, we have $n_{p_i} > M_{p_i}$. Assume that a hitfree sequence exists, say S_{opt}. Let us denote the jobs in S_{opt} whose length does not equal p_i by $i_1, i_2, \ldots, i_L$, where job i_j starts before job i_{j+1} in S_{opt} for $j = 1, \ldots, L-1$. (So there are L jobs in the instance whose length is not p_i.) We will show that given these jobs $i_1, i_2, \ldots, i_L$, the maximal number of jobs of length p_i which can be accommodated without causing a hit in S_{opt} equals M_{p_i}. Thus, since $n_{p_i} > M_{p_i}$, no hitfree sequence can exist.

Since n_{p_i} is finite and since we assumed that $n_{p_i} > M_{p_i}$, it follows that $l_j^{p_i}$ is finite for $j = 1, \ldots, L$. Consider the starting time of job i_1. It is preceded solely by jobs of length p_i. Moreover, it cannot start after $l_0^{p_i} + 1$ jobs of length p_i, because in that case S_{opt} has a hit at time $(l_0^{p_i} + 1)p_i$ (by definition of $l_0^{p_i}$). This implies that we can write the starting time of job i_1 as $(l_0^{p_i} + 1)p_i - \theta_1 p_i$ for some $\theta_1 \geq 1$, $\theta_1 \in \mathbb{N}$. Thus, there are $l_0^{p_i} + 1 - \theta_1$ jobs of length p_i preceding job i_1 in S_{opt}. Further, by definition of $l_{i_1}^{p_i}$, if $l_{i_1}^{p_i} + \theta_1$ jobs of length p_i follow job i_1, we incur a hit at, say, γk for some $\gamma \in \mathbb{N}$. Since we assumed that S_{opt} is hitfree, it follows that the starting time of job i_2 equals $\gamma k - \theta_2 p_i$ for some $\theta_2 \geq 1$, $\theta_2 \in \mathbb{N}$. Thus, the number of jobs of length p_i between job i_1 and job i_2 equals $l_{i_1}^{p_i} + \theta_1 - \theta_2$. More generally, one can argue in a similar fashion that the number of jobs of length p_i between jobs i_j and i_{j+1} equals $l_{i_j}^{p_i} + \theta_j - \theta_{j+1}$ for $j = 1, \ldots, L-1$. Also, the number of jobs of length p_i which can follow job i_L equals $l_{i_L}^{p_i} + \theta_L - 1$ for some $\theta_L \in \mathbb{N}$. Computing the maximal number of jobs of length p_i which can be accommodated in S_{opt} by summing these numbers shows that in S_{opt} maximally $l_0^{p_i} + \sum_j^L l_{i_j}^{p_i} = M_{p_i}$ jobs of length p_i can be sequenced, and thus, since $n_{p_i} > M_{p_i}$, no hitfree sequence exists.

(if) Let us suppose that no hitfree sequence exists. Consider the algorithm. In step 1, the algorithm finds the latest hit in the current sequence, and by exchanging the job starting at that hit with some job in the sequence before the hit with a different length avoids the latest hit without inducing later hits. By repeating step 1, either a sequence with safety distance 1 is constructed (but this is impossible by assumption)

or the following situation appears. Let the sequence resulting from step 1 be denoted by S. All jobs in S sequenced before the latest hit and the one starting at this hit have equal length, say x. Let the jobs in S whose length is not equal to x be denoted by $j_1, j_2 \ldots, j_N$, where job j_i starts before job j_{i+1} in S for $i = 1, \ldots, N-1$. (So there are N jobs in the instance whose length is not equal to x.) Notice further that the current sequence S has the property that all hits are "caused" by jobs of length x, that is, each job ending at a hit has length x.

We will now argue that this property is preserved in all iterations of step 3. In the first iteration of step 3, job j_1 is used to avoid the first hit. Obviously, since there are no hits after the original position of job j_1 (all jobs before the last hit have length x), placing j_1 at the first hit will not induce hits after the original position of j_1. Thus, after the first iteration, all hits in the sequence will be present between the positions of jobs j_1 and j_2, and are caused by jobs of length x. In general, placing job j_i at the first hit in the current sequence will result in a situation where all hits are caused by jobs of length x sequenced between jobs j_i and j_{i+1} for some $1 \leq i \leq N-1$. Therefore, assuming that no hitfree sequence exists, the algorithm will end up with a sequence where all hits are caused by jobs of length x. Where can the first hit be? We know that if a hit results by placing job j_i at $qk - x$ for some $q \in \mathbb{N}$ (as the algorithm does), job j_i must be followed by $l^x_{j_i} = 1$ jobs of length x. Then step 2 proceeds by placing job j_{i+1} following the $l^x_{j_i}$ jobs of length x, which follow job j_i. Since this holds for $i = 1, \ldots, N-1$, it is clear that the first hit can only occur after $l^x_0 + \sum_{i=1}^{N} l^x_{j_i} + 1$ jobs of length x. Hence, $n_x > M_x$, and the lemma follows. □

The complexity of the algorithm is determined by step 1, where for each possible hit (of which there can be no more than $O(n)$), $O(n)$ jobs have to be investigated. This gives an overall running time of $O(n^2)$.

In the case that a hit is unavoidable, the algorithm produces a sequence in which the first hit is as late as possible. This follows from the proof and from the fact that in any instance there can be at most one job length x with $n_x > M_x$.

THEOREM 3.5

If $d = 1$ and if all jobs in J_1 have equal length, the problem SSD can be solved in $O(n^2)$ time.

Proof

In case conditions (C1), (C2) and (C3) are fulfilled, the claim follows from lemma 3.4 and the above discussion.

If (C2) is not fulfilled, no feasible schedule can exist, since the two final jobs always collide. If (C1) is not fulfilled (and (C3) is fulfilled), we apply the algorithm for detecting a hitfree sequence. In case a hit is unavoidable, we get a sequence in which the first hit is as late as possible. If the first hit occurs after P_1, we have found a feasible schedule, otherwise no solution exists.

Finally, suppose condition (C3) is not fulfilled. Then we replace all jobs in J_2 by their remainders modulo k. This gives a new job set J_2'. This new instance may contain pseudo-jobs of length zero (ordinary jobs must have positive length). If all jobs in J_2' are of length zero, a feasible solution exists if and only if the largest job in J_2 has a length $> P_1$. If not all jobs in J_2' are of length zero, we remove the pseudo-jobs from J_2' and apply our algorithm for detecting a hitfree sequence (so we assume that all completion times of the form ik are forbidden, irrespective of the value of P_1). (Obviously, if the algorithm finds a hitfree sequence for J_2', we can easily construct a hitfree sequence for the original instance, by simply adding all pseudo-jobs to the end of the current sequence and by giving the jobs their original length.) In case a hit is unavoidable, we found a sequence in which the first hit is as late as possible. Moreover, the algorithm ensures that each hit in this sequence is caused by a job of a certain length, say x, and that all jobs of length not x (if any) are sequenced before the first hit. In the following way, we will convert this solution to a solution for the original job set J_2, in which the first job with a completion time of the form ik for some i is as late as possible. Then, by checking whether $P_1 < ik$, it is easy to see whether a hitfree sequence exists.

To convert the solution for J_2' to a solution for J_2, we are going to give the jobs their original length. First, we insert all pseudo-jobs somewhere before the first hit, say between the first and the second job in the current sequence. Second, we give all jobs whose length is not x their original length (notice that these jobs all occur before the first hit). Third, order the original lengths of the jobs of length x in J_2' in a nonincreasing manner, say $r_1 \geq r_2 \geq \ldots$. Then, by giving the ith job of length x in the current sequence length r_i, we maximize the time unit at which the first potential hit may occur. Of course, one might wonder whether another sequence of the jobs in J_2' might allow for a larger delay of the first potential hit. This, however, is impossible, since any other sequence of the jobs in J_2' has fewer jobs of length x before the first hit (see the proof of lemma 3.4). So, checking if $P_1 < ik$ decides whether a hitfree sequence exists. □

Until now, in this subsection we dealt with a variant of SSD which has as input $n + 1$ numbers, namely the n processing times of the jobs in J_2 together with the number k. Let us now consider a variant where the input is more compact. More specifically, suppose the input consists of the number k and, for $i = 1,\ldots,D$, p_i and n_{p_i}, that is, only $O(D)$ numbers. Further, let us define for $i, j = 1,\ldots,D$, $l_{p_j}^{p_i}$ as the maximal number of jobs of length p_i which, when sequenced consecutively following a job with length p_j which starts at time unit $-p_i$, do not induce a hit. Problems where we can partition the jobs into relatively few classes such that the characteristics of all jobs belonging to a class are identical are called *high-multiplicity problems* in Hochbaum and Shamir [9]. The following remark applies to this high-multiplicity case.

Remark 3.6

The fact that we can compute $l_{p_j}^{p_i}$ by the Euclidean algorithm implies for the high-multiplicity variant of SSD that we can decide in polynomial time by applying lemma 3.4 whether a hitfree schedule exists.

4. Discussion

We introduced a scheduling problem called SSD whose main goal essentially consists in avoiding job completion times that are too close to each other. We proved this problem to be NP-complete, and we solved several special cases in polynomial time. A sharp borderline between NP-completeness and polynomial solvability of SSD was established.

There remain several questions to be investigated:

(1) In the corresponding optimization problem, one wants to find a schedule that *maximizes* the safety distance. It would be interesting to have an approximation algorithm with reasonable worst-case guarantee. (Since it is NP-complete to decide whether a schedule with safety distance $d = 1$ exists, the best possible result would be of the form $d^H \geq c_1 d^* - c_2$, where c_1 and c_2 are positive reals, d^H is the safety distance produced by the heuristic, and d^* is the optimum safety distance. Of course, c_1 should be as large as possible.)

(2) We conjecture that if there is only a constant number c of distinct job lengths, then there exists a polynomial time algorithm for SSD with running time $O(n^{f(c)})$.

Acknowledgements

We would like to thank B. Fruhwirth and J. van de Klundert for several helpful discussions and for providing some pointers to the literature.

References

[1] A. Agnetis, F. Nicolo and M. Lucertini, Tool handling synchronization in flexible manufacturing cells, *Proc. 1991 IEEE Int. Conf. on Robotics and Automation* (1991) pp. 1789–1794.

[2] P. Brucker and W. Meyer, Scheduling two irregular polygons, Discr. Appl. Math. 20(1988) 91–100.

[3] R.E. Burkard, Optimal schedules for periodically recurring events, Discr. Appl. Math. 15(1986) 167–180.

[4] P. Brucker, R.E. Burkard and J. Hurink, Cyclic schedules for r irregularly occurring events, J. Comp. Appl. Math. 30(1990)173–189.

[5] B. Fruhwirth, private communication (1992).

[6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979).

[7] F. Guldan, Maximization of distances of regular polygons on a circle, Aplikace Matematiky 25(1980)182–195.
[8] N.G. Hall, C.N. Potts and C. Sriskandarajah, Parallel machine scheduling with a common server, Research Report (1994).
[9] D.S. Hochbaum and R. Shamir, Strongly polynomial algorithms for the high multiplicity scheduling problem, Oper. Res. 8(1991)648–653.
[10] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization* (Wiley, New York, 1988).