Central European Journal of Operations Research
https://doi.org/10.1007/s10100-022-00803-7

On the minimum number of resources for a perfect schedule

Rachid Benmansour[1] · **Oliver Braun**[2]

Accepted: 29 April 2022

Abstract
In the single-processor scheduling problem with time restrictions there is one main processor and B resources that are used to execute the jobs. A *perfect schedule* has no idle times or gaps on the main processor and the makespan is therefore equal to the sum of the processing times. In general, more resources result in smaller makespans, and as it is in practical applications often more economic not to mobilize resources that will be unnecessary and expensive, we investigate in this paper the problem to find the smallest number B of resources that make a perfect schedule possible. We show that the decision version of this problem is NP-complete, derive new structural properties of perfect schedules, and we describe a Mixed Integer Linear Programming (MIP) formulation to solve the problem. A large number of computational tests show that (for our randomly chosen problem instances) only $B = 3$ or $B = 4$ resources are sufficient for a perfect schedule.

Keywords Minimizing the number of resources · Perfect schedule · Single-processor scheduling · Mixed integer linear programming

1 Introduction

In the single-processor scheduling problem with time restrictions (STR), there are n independent jobs $J_1, \ldots, J_n$ (or $1, \ldots, n$) with positive integer processing times (or job lengths) s_j, $j \in \{1, 2, \ldots, n\}$. The jobs have to be processed non-preemptively on a single processor (note that the processor could be a computer processor or a person/worker who operates resources/machines). A feasible schedule is a permutation

✉ Oliver Braun
o.braun@umwelt-campus.de

Rachid Benmansour
r.benmansour@insea.ac.ma

[1] SI2M Laboratory, Institut National de Statistique et d'Économie Appliquée, Rabat, Morocco

[2] Trier University of Applied Sciences, Environmental Campus Birkenfeld, 55768 Birkenfeld, Germany

Published online: 26 May 2022

$\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ of the jobs with corresponding processing times $p_1, p_2, \ldots, p_n$, completion times $C_1, C_2, \ldots, C_n$, makespan $C_{max} = C_n$ and the following property: The initial job π_1 starts at time 0 and completes its processing at time p_1. For $k \geq 2$, job π_k starts no earlier than job π_{k-1} has completed its processing, and possibly later as the following constraint must always be satisfied:

> Each job requires the use of one of B identical additional resources that have to be renewed in α time units after the processing of a job has been finished and before they can be used again (Braun et al. 2014).

Because of its general formulation, this model is widely applicable. The renewal time of the resource reflects that it has to be cleaned, transported to another place, cool down, refilled, re-loaded, updated, etc. A practical application can be found in a production environment in which one main machine can use several resources that must be cleaned etc. after their usage. Another application would be, for example, one or more team members who can work on a project for a certain amount of time and then are locked out until they can be reassigned to a new project.

From the constraint above it follows that *1. At most B jobs can be processed during any interval* $[x, x + \alpha) \forall x \in \mathbb{R}_{\geq 0}$, and that 2.$\forall x \in \mathbb{R}_{\geq 0}$, *the interval* $[x, x + \alpha)$ *can intersect at most B jobs.*

The following example with $n = 4$ jobs J_1, J_2, J_3, J_4, processing times $s_1 = 6, s_2 = 4, s_3 = 2, s_4 = 4$ and renewal times $\alpha = 10$ is given for better illustration. Figures 1 and 2 present two feasible schedules for $B = 2$. As an example, in Fig. 1, resource R_1 is occupied by job J_1 from timepoint 0 to timepoint 6 and has to be renewed afterwards which takes 10 timeunits until timepoint 16. In order to schedule job J_2 immediately after J_1 we need another resource R_2 which is occupied by J_2 from timepoints 6 to 10 and which has to be renewed afterwards from timepoint 10 to timepoint 20. It follows that we have a gap on the main processor from timepoint 10 until timepoint 16. The schedule (1, 2, 3, 4) is not optimal with respect to minimzing the makespan, whereas the schedule (3, 2, 1, 4) is an optimal schedule with makespan $C^*_{max} = 22$ (on the main processor).

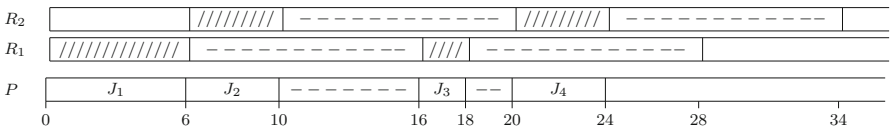

Fig. 1 Schedule (1, 2, 3, 4) with $n = 4$ jobs, $B = 2$ resources R_1, R_2, renewal times $\alpha = 10$, and makespan $C_{max} = 24$

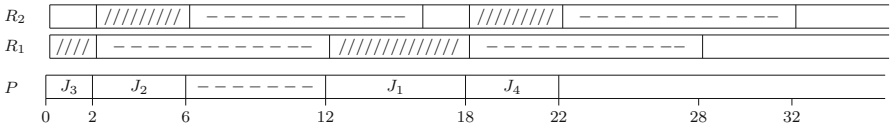

Fig. 2 Schedule (3, 2, 1, 4) with $n = 4$ jobs, $B = 2$ resources R_1, R_2, renewal times $\alpha = 10$, and optimal makespan $C^*_{max} = 22$

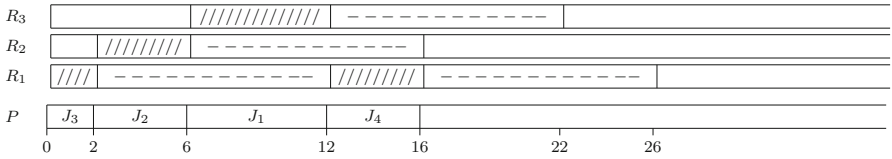

Fig. 3 Perfect schedule $(3, 2, 1, 4)$ with $n = 4$ jobs, $B = 3$ resources R_1, R_2, R_3, renewal times $\alpha = 10$, and optimal makespan $C^*_{max} = \sum_{j=1}^{4} s_j = 16$

It is obvious that in the STR-problem more resources lead to smaller (or at least not larger) makespans and that never more than n resources are necessary to schedule the jobs on the main processor without any idle times (or gaps). In this paper, we investigate the question how many resources are needed at most (in the sense of a supremum) for such a *perfect schedule* (Braun et al. 2014). We call this problem the *STR-B-problem*. The idea behind this problem is that in practical applications it is often more economic not to mobilize resources that will be unnecessary and expensive (Rustogi and Strusevich 2013). This kind of question arises as well in scheduling problems with no-wait constraints, e.g. Ruiz et al. (2009).

To continue with our introductory example, it turns out that only $B = 3$ resources are necessary for a perfect schedule $(3, 2, 1, 4)$ with makespan $C^*_{max} = \sum_{j=1}^{n} s_j = 16$ (Fig. 3).

Another example with $n = 10$ jobs, processing times 7, 14, 19, 25, 27, 31, 38, 38, 49, 71 and renewal times $\alpha = 100$ has the following optimal makespan values for a different number of resources: $C^*_{max} = 570$ $(B = 2)$, $C^*_{max} = 373$ $(B = 3)$, $C^*_{max} = 319$ $(B = 4)$. The optimal makespan values decrease with an increasing number B of resources until C^*_{max} reaches for $B = 4$ the value of the sum of the processing times $\sum_{j=1}^{10} s_j = 319$, i.e. for $B = 4$ there is a perfect schedule that has no gap (or delay) on the main processor.

The single-processor scheduling problem with time restrictions (STR) was at first studied by Braun et al. (2014, 2016). The authors show that the decision version of the STR-problem is NP-complete when the number of resources B is part of the input and therefore possibly arbitrarily large. They analyze the worst-case behaviour of *List Scheduling* (where the jobs are scheduled in an arbitrary permutation) and prove that for $B = 2$ the best possible worst-case factor of *List Scheduling* is $\frac{4}{3}$ of the optimum (plus the additional constant 1), and that for $B \geq 3$, the best possible worst-case factor is equal to $2 - \frac{1}{B-1}$ of the optimum (plus the additional constant $B/(B-1)$). Moreover, the authors analyze the *Longest-Processing-Time-first (LPT)*-algorithm, where the jobs are ordered non-increasingly and show that LPT-ordered jobs can be processed within the best possible factor of $2 - 2/B$ of the optimum (plus the additional constant $\frac{1}{2}$ for $B = 2$ and 1 for $B \geq 3$). Zhang et al. (2017) show independently the same bound for $B = 2$. Moreover, they provide an approximation algorithm for $B \geq 3$ that achieves the factor $\frac{3}{2}$ plus the additional constant 2 for $B = 3$, the factor $\frac{4}{3}$ plus the additional constant 2 for $B = 4$, and the factor $\frac{5}{4}$ plus the additional constant 2 for $B \geq 5$. Zhang et al. (2017) prove that the decision version of the STR-problem is even NP-complete

for $B = 2$ and they describe a Polynomial Time Approximation Scheme (PTAS) for any fixed value $B \geq 2$.

Benmansour et al. (2018) propose Mixed Integer Linear Programming (MIP) formulations, based on a time-indexed formulation and based on an assignment and positional date formulation, to solve the STR-problem and they prove that the decision version of the STR-problem is NP-complete even for $B = 2$. Benmansour et al. (2019) present two algorithms, namely Variable Neighborhood Search (VNS) and Fixed Neighborhood Search (FNS), for the approximate solution of the STR-problem.

There is an interesting connection between the single-processor scheduling problem with time restrictions and the parallel machine scheduling problem with a single server (PSS, $P, S1 \mid s_i, p_i' \mid C_{max}$) (Benmansour et al. 2018; Kravchenko and Werner 1997). In PSS, s_i is the setup time to load a job i on a common server, and p_i' is the processing time of that job. The server and the processor are both occupied during the loading operation. STR and PSS are in fact equivalent problems: The setup times of PSS are equal to the processing times of STR (it might well be that in a practical application some jobs need more or less time than others to be ready for being processed), and the processing times of PSS are equal to the renewal times of STR (which is a constant α in this case). Therefore, our analysis gives also an answer to the following question: *How many parallel machines do we need at least to construct a schedule of the jobs that has no idle times on the single server?*

The remainder of the paper is organized as follows. In Sect. 2 we prove that the decision version of STR-B is NP-complete by reducing the decision version of (the known NP-complete problem) STR to the decision version of STR-B. In Sect. 3, we develop structural properties of perfect schedules, and we present a Mixed Integer Programming (MIP) formulation to solve the STR-B problem. Section 4 presents computational performance tests of the MIP. Finally, in Sect. 5 we give a conclusion.

2 The decision version of STR-B is NP-complete

It is easy to see that in the worst-case, there must be n resources available in order to schedule the jobs without gaps on the main processor. As an example: When the sum of the $n - 1$ largest processing times is less than α, then $n - 1$ resources are not sufficient and we need as many as n resources for a perfect schedule. From a computational complexity point of view, STR-B is not harder than STR as STR-B can be solved by solving at most $\lfloor \log n \rfloor + 1$ instances of the STR-problem: We just have to perform a binary search to determine the smallest B such that the makespan of the solution to the STR-problem is equal to the sum of the processing times. Note that the optimal makespans for an increasing number B of resources are non-increasing.

Theorem 1 *The decision version of the STR-B-problem (given n processing times s_j, $j = 1, \ldots, n$, with $S = \sum_{j=1}^{n} s_j$, the renewal time α of the resources, and a number B of resources, is there a feasible schedule with a makespan $C_{max} = S$?) is NP-complete.*

Proof STR-B is obviously in NP: Given B and a schedule π, i.e. a permutation of the jobs, it is in polynomial time possible to check that the jobs can be scheduled

without gaps on the main processor using at most B resources. The decision version of the STR-problem is as follows: Given n processing times s_j, $j = 1, \ldots, n$, the number B of resources, the renewal time α of the resources, and a makespan C_{max}, is there a feasible schedule with a makespan not larger than C_{max}? It is known to be NP-complete (Benmansour et al. 2018). We want to show that STR $\propto$ STR-B: Given an instance of STR-B, we ask if it is possible to schedule the jobs without any gap on the main processor with B resources. The only way to answer this question is to solve STR. Conversely, given a solution to STR that uses B resources and has no gaps on the main processor yields immediately to a solution to STR-B (we would possibly and in the worst-case have to solve $\lfloor \log n \rfloor + 1$ instances of the STR-problem as described above). Since STR-B is in NP, since the input for STR-B can be computed in polynomial time from the input for STR, and since we can reduce the NP-complete problem STR to STR-B, STR-B must also be NP-complete. □

As a remark, Braun et al. (2014) show that the decision version of STR (when the value B is variable) is NP-complete through a reduction of PARTITION to the special case of STR where there is a perfect schedule.

3 MIP formulation

We start with a useful property of perfect schedules.

Theorem 2 *There is always a perfect schedule where the two jobs with the smallest processing times are scheduled at the beginning and at the end of the schedule.*

Proof We assume w.l.o.g. that J_{n-1} and J_n are the jobs with the smallest processing times s_{n-1} and s_n and claim that there is always a perfect schedule with a permutation $(n-1, \pi_2, \ldots, \pi_{n-1}, n)$. Imagine a perfect schedule π' where the first $B-1$ jobs have a sum of processing times $\geq \alpha$ and use only $B-1$ resources. We can construct another perfect schedule π by using the B^{th} resource for processing another job in the beginning. This resource will be available again after α time units. Therefore scheduling this new job at the very beginning will not cause any delay. The same is true for the last $B-1$ jobs in a perfect schedule. Again, we can use the Bth resource for another job. It follows that there is always a perfect schedule where the two smallest jobs are scheduled at the beginning and at the end of the schedule. □

Next we describe a necessary and sufficient condition for a perfect schedule with B resources.

Theorem 3 *Necessary and sufficient conditions for a perfect schedule* $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ *with processing times* $p_1, p_2, \ldots, p_n$ *and* B *resources are:*

$$\sum_{k=i}^{i+B-2} p_k \geq \alpha \quad \forall i \in \{2, \ldots, n-(B-1)\} \tag{1}$$

Proof If $C_i + \alpha$ would be greater than $C_{i+(B-1)}$, then there would be a gap in the schedule. This observation leads to the following necessary conditions for a perfect schedule: $C_i + \alpha \leq C_{i+(B-1)} \quad \forall i \in \{2, \ldots, n-(B-1)\}$. From this it follows immediately that the following constraints must be satisfied in a perfect schedule: $\sum_{k=i}^{i+B-2} p_k \geq \alpha \quad \forall i \in \{2, \ldots, n-(B-1)\}$. □

As an example, consider the situation for $B = 4$, $\alpha = 1000$ and $n = 10$ jobs with $p_4 = p_7 = 1000$ and $p_j = 1$ for all other jobs. This schedule is a feasible schedule and we have

$$\begin{aligned} p_1 + p_2 + p_3 &< \alpha, \\ p_2 + p_3 + p_4 &\geq \alpha, \\ p_3 + p_4 + p_5 &\geq \alpha, \\ p_4 + p_5 + p_6 &\geq \alpha, \\ p_5 + p_6 + p_7 &\geq \alpha, \\ p_7 + p_8 + p_9 &\geq \alpha, \\ p_8 + p_9 + p_{10} &< \alpha. \end{aligned}$$

Note that $p_1 + p_2 + p_3$ and $p_8 + p_9 + p_{10}$ can be smaller than α as the corresponding jobs are scheduled at the beginning and at the end of the schedule.

Another useful observation is about the number of jobs that must have a certain length to build a perfect schedule.

Theorem 4 *In a perfect schedule $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ with processing times $p_1, p_2, \ldots, p_n$ and B resources, at least $\left\lceil \frac{n-B}{B-1} \right\rceil$ jobs have processing times $p_j \geq \frac{\alpha}{B-1}$.*

Proof Let J_{n-1} and J_n be the two jobs with the smallest processing times s_{n-1} and s_n. We observe that in a perfect schedule π (where we put J_{n-1} in the front and J_n at the end of the schedule), all of the other jobs must fulfill the following property: Always $B-1$ processing times of adjacent jobs have to sum up to at least α. In a perfect schedule for B resources, by the pigeonhole principle, in each of the inequations (1) from Theorem 3, at least one job has to have a processing time that is $\geq \frac{\alpha}{B-1}$. As we can schedule the two smallest jobs at the beginning and at the end of a perfect schedule (Theorem 2), it remains that in a perfect schedule at least $\left\lceil \frac{n-B}{B-1} \right\rceil$ jobs have processing times $\geq \frac{\alpha}{B-1}$. □

In our example for Theorem 3, we see that there are $\left\lceil \frac{n-B}{B-1} \right\rceil = \left\lceil \frac{10-4}{4-1} \right\rceil = 2$ jobs, namely p_4 and p_7 with processing times $p_j \geq \frac{\alpha}{B-1} = \frac{1000}{3}$.

As a result, before starting the MIP, we check if the necessary conditions from Theorem 4 for a perfect schedule for $B = 2$ are fulfilled. If yes, this gives the lower bound l on the number of resources that are needed. Otherwise, we increase B by 1 and continue until we found a lower bound (Algorithm 1).

An upper bound is obviously $u = n$ as there might be as many resources as there are jobs necessary to construct a perfect schedule (see the example in Sect. 2). However,

Algorithm 1 Algorithm to determine a Lower Bound l on the number of resources for a perfect schedule.

procedure LOWERBOUND(n, $p_1, \ldots, p_n, \alpha$)
  $l := 2$;
  while number of jobs with processing times $\geq \frac{\alpha}{l-1}$ is smaller than $\left\lceil \frac{n-l}{l-1} \right\rceil$ **do**
    $l := l + 1$;

there might be tighter upper bounds possible. We did not investigate this question further, but we decided to use u as a parameter in the MIP. The resulting MIP formulation to solve the STR-B-problem is given in Fig. 4.

$$\min \quad B = \sum_{b=l}^{u} b y_b \tag{1}$$

s.t.

$$\sum_{b=l}^{u} y_b = 1 \tag{2}$$

$$\sum_{k=2}^{n-1} x_{jk} = 1 \quad \forall j \in \{1, \ldots, n-2\} \tag{3}$$

$$\sum_{j=1}^{n-2} x_{jk} = 1 \quad \forall k \in \{2, \ldots, n-1\} \tag{4}$$

$$\sum_{k=i}^{i+b-2} \sum_{j=1}^{n-2} s_j z_{jk}^b \geq \alpha y_b \quad \forall b \in \{l, \ldots, u\}, \forall i \in \{2, \ldots, n-(b-1)\} \tag{5}$$

$$z_{jk}^b \leq x_{jk} \quad \forall b \in \{l, \ldots, u\}, \forall j \in \{1, \ldots, n-2\}, \forall k \in \{2, \ldots, n-1\} \tag{6}$$

$$z_{jk}^b \leq y_b \quad \forall b \in \{l, \ldots, u\}, \forall j \in \{1, \ldots, n-2\}, \forall k \in \{2, \ldots, n-1\} \tag{7}$$

$$z_{jk}^b \geq x_{jk} + y_b - 1 \quad \forall b \in \{l, \ldots, u\}, \forall j \in \{1, \ldots, n-2\}, \forall k \in \{2, \ldots, n-1\} \tag{8}$$

$$y_b, x_{jk}, z_{jk}^b \in \{0, 1\} \quad \forall b \in \{l, \ldots, u\}, \forall j \in \{1, \ldots, n-2\}, \forall k \in \{2, \ldots, n-1\} \tag{9}$$

Fig. 4 MIP to determine the minimum number of resources for a perfect schedule

The objective function (1) minimizes the number of B of resources that are necessary for a perfect schedule. Since the value of B is not known in advance, we introduce the binary variables y_b such that $B = \sum_{b=l}^{u} b y_b$, where l is the lower bound (determined by Algorithm 1) and u is the upper bound (we chose $u = n$) on the number of resources that are needed for a perfect schedule. Constraint (2), $\sum_{b=l}^{u} y_b = 1$, assures that B equals exactly one value (the minimum number of resources that are necessary to find a perfect schedule) out of the possible values $\{l, \ldots, u\}$. It assures that exactly one of the binary variables $y_l, \ldots, y_u$ is equal to 1.

The binary variable x_{jk} corresponds to the assignment of job j to position k (i.e., $x_{jk} = 1$ if and only if job j is assigned to position k). Note that in a perfect schedule, we can place the two jobs with the smallest processing times (w.l.o.g. J_{n-1} and J_n with processing times s_{n-1} and s_n) to positions $k = 1$ and $k = n$ so that $\pi_1 = n - 1$ and $\pi_n = n$ with $p_1 = s_{n-1}$ and $p_n = s_n$ (Theorem 2). We then have to decide at what positions $2, \ldots, n-1$ to place jobs $J_1, \ldots, J_{n-2}$ in the optimal permutation π. Therefore the job index variable j always runs from 1 to $n-2$ and the position variable k always runs from 2 to $n-1$.

Constraints (3) and (4) state that each job is assigned to only one position and that each position is assigned to exactly one job.

The constraints from inequalities (1) of Theorem 3

$$\sum_{k=i}^{i+B-2} p_k \geq \alpha \quad \forall i \in \{2, \ldots, n-(B-1)\}$$

are reflected in constraints (5).

Note that

$$p_k = \sum_{j=1}^{n-2} s_j x_{jk}$$

is the processing time of job j at position k and that the binary variable $x_{jk} = 1$ if and only if job j is assigned to position k.

Constraints (6–8) model the linearization of the product of the two binary variables x_{jk} and y_b:

$$z^b_{jk} = x_{jk} y_b$$

Constraints (6) and (7) ensure that z^b_{jk} will be zero if either x_{jk} or y_b are zero. Constraints (8) make sure that z^b_{jk} will take value 1 if both binary variables x_{jk} and y_b are one.

The variables y_b, x_{jk}, z^b_{jk} are defined as binary variables in constraints (9).

4 Computational tests

In this section, we describe the results of the computational tests for the MIP. The meaning of the parameters are explained in Table 1. We used an Intel i7 1.8 GHz processor with 16 GB RAM and IBM ILog CPLEX 20.10 using default settings. It is obvious that the number of resources for a perfect schedule are smaller when α is small in comparison to the processing times. Therefore, we restricted the processing times of the jobs to be generated from a discrete uniform distribution in $[1, \alpha]$, i.e. no job has a processing time that is larger than the renewal time α of the resources.

Table 1 Parameters for the computational tests

n	Number of jobs, $n = 10, 50, 100$
α	Renewal time of the resources, $\alpha = 10, 100, 1000$
s_j	(Integer) processing time of job J_j, $1 \leq s_j \leq \alpha$
$\sum s_j$	Sum of the processing times $\sum s_j = \sum_{j=1}^{n} s_j$
Lower Bound	Lower bound for the number of resources calculated by Algorithm 1
B (MIP)	Number of resources obtained by the MIP; an asterisk * in the B-column indicates that B is optimal because the MIP provably found the optimal solution
Runtime (in seconds)	Runtime of the MIP (in seconds); we stopped the MIP after at most 3000 s

4.1 n = 10 jobs

The computational results for $n = 10$ jobs and $\alpha = 10, 100, 1000$ are displayed in Table 2.

The MIP found the optimal solutions for all of the problem instances in less than 0.1 seconds. In 4 out of the 30 problem instances the optimal MIP result is not equal to the lower bounds calculated by Algorithm 1.

4.2 n = 50 jobs

In Table 3 the results for $n = 50$ jobs and $\alpha = 10, 100, 1000$ are displayed.

All of the problem instances with $\alpha = 10, 100$ could be optimally solved by the MIP. In 5 out of 20 problem instances the optimal MIP solution was not equal to the lower bound. The hardest problem instances for $n = 50$ jobs were those with $\alpha = 1000$. All of the 4 problem instances with a lower bound of 4 could be solved optimally by the MIP in less than 0.1 seconds. For two problem instances where the lower bound is only 3, the MIP could only find a solution with $B = 4$ resources in the given time (3000 s).

4.3 n = 100 jobs

In Table 4 the results for $n = 100$ jobs and $\alpha = 10, 100, 1000$ are displayed.

10 out of the 30 problem instances could not be solved provable optimally by the MIP. While all of the problem instances with a lower bound 4 could be optimally solved by the MIP, there are some problem instances with a lower bound 3 where the MIP could only find a solution with $B = 4$ resources.

We find it interesting that our MIP could not find the optimal solution for e.g. problem instance 7 of $n = 100$, $\alpha = 100$. The result of the MIP is $B = 4$, but there are perfect schedules possible for this problem instance with only $B = 3$ resources: Take out the two smallest jobs, sort the remaining jobs from large to small (i.e. $s_2 \geq s_3 \geq$

Table 2 Results for $n = 10$ jobs

	$\sum s_j$	Lower bound	B (MIP)	Runtime (in s)
$n = 10, \alpha = 10$				
1	61	3	3*	0.01
2	46	3	3*	0.03
3	50	3	3*	0.02
4	50	3	3*	0.02
5	56	3	3*	0.02
6	47	3	3*	0.02
7	53	3	3*	0.02
8	43	4	4*	0.08
9	56	3	3*	0.01
10	44	3	4*	0.02
$n = 10, \alpha = 100$				
1	564	3	3*	0.03
2	414	3	4*	0.02
3	457	3	3*	0.02
4	394	3	4*	0.03
5	650	3	3*	0.01
6	457	3	4*	0.11
7	519	3	3*	0.02
8	571	3	3*	0.02
9	319	4	4*	0.03
10	439	4	4*	0.06
$n = 10, \alpha = 1000$				
1	5541	3	3*	0.01
2	4746	3	3*	0.01
3	6506	3	3*	0.01
4	4482	4	4*	0.02
5	6122	3	3*	0.02
6	6598	3	3*	0.01
7	5566	3	3*	0.02
8	5506	3	3*	0.01
9	5481	3	3*	0.01
10	3825	4	4*	0.02

s_{99}), then schedule always a large job next to a small job, so that the final schedule is $(1, 2, 99, 3, 98, \ldots, 50, 51, 100)$. This *alternating schedule* is obviously promising as a heuristic or even an approximation algorithm for $B = 3$ resources.

We note that our implementation of a *Variable Neighborhood Search* (VNS) heuristic lead in no case to a smaller number B of resources for a perfect schedule. This is why we decided not to present the VNS results in this paper.

Table 3 Results for $n = 50$ jobs

	$\sum s_j$	Lower bound	B (MIP)	Runtime (in s)
$n = 50, \alpha = 10$				
1	287	3	3*	1.58
2	256	3	3*	19.41
3	276	3	3*	2.20
4	262	3	3*	7.61
5	277	3	3*	2.61
6	267	3	3*	3.59
7	266	3	3*	3.48
8	269	3	3*	3.69
9	276	3	3*	3.30
10	235	3	4*	0.75
$n = 50, \alpha = 100$				
1	2646	3	3*	8.27
2	2590	3	3*	235.03
3	2460	4	4*	0.03
4	2833	3	3*	1.77
5	2612	3	4*	443.28
6	2387	4	4*	0.08
7	2553	3	4*	36.66
8	2408	3	4*	11.02
9	2316	4	4*	0.06
10	2569	3	4*	38.39
$n = 50, \alpha = 1000$				
1	25,591	3	3*	1182.61
2	25,015	3	4	3000.00
3	25,836	3	3*	734.09
4	23,098	4	4*	0.06
5	24,602	4	4*	0.03
6	27,772	3	3*	5.77
7	25v420	3	4*	6.86
8	22,443	4	4*	0.06
9	22,142	4	4*	0.08
10	25,055	3	4	3000.00

We note further that we tested *List Scheduling* (where the jobs are given in a randomly chosen permutation) and *LPT* (where the jobs are sorted in non-increasing order of their processing times) on all of the problem instances but that these algorithms were not effective at all (with *LPT* as expected even worse than *List Scheduling*).

From a practical point of view the MIP performs very well. Only 14 out of 90 problem instances could not be provably optimally solved by the MIP. In all of these

Table 4 Results for $n = 100$ jobs

	$\sum s_j$	Lower bound	B (MIP)	Runtime (in s)
$n = 100, \alpha = 10$				
1	584	3	3*	12.89
2	489	3	4*	3.78
3	569	3	3*	17.28
4	565	3	3*	22.14
5	510	3	3*	741.95
6	541	3	3*	26.99
7	560	3	3*	20.80
8	527	3	3*	104.28
9	567	3	3*	17.78
10	545	3	3*	38.58
$n = 100, \alpha = 100$				
1	5317	3	4	3000.00
2	5025	4	4*	0.23
3	5047	4	4*	0.13
4	4886	4	4*	0.20
5	5365	3	4	3000.00
6	5120	3	4*	568.30
7	5250	3	4	3000.00
8	5255	3	4	3000.00
9	5138	3	4	3000.00
10	4906	3	4	3000.00
$n = 100, \alpha = 1000$				
1	49,664	3	4	3000.00
2	46,447	4	4*	0.22
3	49,099	4	4*	0.20
4	55,427	3	3*	50.45
5	54,190	3	4	3000.00
6	53,146	3	4	3000.00
7	56,018	3	3*	24.00
8	50,177	3	4	3000.00
9	47,753	3	4*	4.70
10	50,001	4	4*	0.22

cases the number of resources achieved by the MIP is 4 and the lower bound on the minimum number of resources necessary for a perfect schedule is 3.

5 Conclusion

We introduced a scheduling problem where the objective is to find the minimum number of external resources in order to find a perfect schedule, i.e. a schedule of

the jobs that has no idle times or gaps on the main processor. We showed that the decision version of this problem is NP-complete, derived new structural properties of perfect schedules, described a MIP formulation, and performed computational tests. We observed that for all problem instances either $B = 3$ or $B = 4$ resources are necessary for a perfect schedule. As we chose the processing times of the jobs from discrete uniform distributions in $[1, \alpha]$, the expected processing time of a job is $(\alpha + 1)/2$. Though possible, it is very unlikely that $n - 2$ out of the n jobs have processing times that are equal to α. But as a result from Theorem 4, this would be a necessary condition that only $B = 2$ resources are sufficient for a perfect schedule. On the other hand, if $B = 3$, then we need at least $\lceil \frac{n-3}{2} \rceil$ jobs with processing times that are at least $\alpha/2$ and this is indeed what we can expect. For $B = 4$ we need at least $\lceil \frac{n-4}{3} \rceil$ jobs with processing times of at least $\alpha/3$ and this is highly probable. Of course, these are only necessary conditions but as the computational tests show that for all of our problem instances either $B = 3$ or $B = 4$ resources are sufficient for a perfect schedule. Note that this implies that for randomly generated problem instances any schedule is optimal if we allow at least $B = 4$ resources. This might be a valuable hint from a managerial perspective.

The worst-case bounds of Braun et al. (2014, 2016) for arbitrary schedules (i.e. permutations of jobs) achieve asymptotically a worst-case-factor of even 2 for the relation between the makespans of arbitrary schedules and optimal schedules. In more detail, the asymptotic worst-case factors are $\frac{4}{3}$ for $B = 2$, $\frac{3}{2}$ for $B = 3$, $\frac{5}{3}$ for $B = 4$, and 2 for $B \to \infty$. This is another example for the well-known observation that often worst-case factors might be too pessimistic for arbitrary problem instances. Another observation, related to the result of Rustogi and Strusevich (2013) is that in the single-processor scheduling problem with time restrictions more resources do not necessarily help. Again, at most $B = 4$ resources are sufficient for perfect schedules. Finally, despite the single-processor scheduling problem with time restrictions is NP-hard for the number of resources B being a variable parameter of the problem (Braun et al. 2014) and even for $B = 2$ (Zhang et al. 2017), the single-processor scheduling problem with time restrictions is easily solvable for our randomly chosen problem instances by any permutation of the jobs when there are $B = 4$ or more resources.

Acknowledgements We thank two anonymous referees for valuable comments that helped to improve the paper.

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Benmansour R, Braun O, Hanafi S (2018) The single processor scheduling problem with time restrictions: complexity and related problems. J Sched 22:465–471

Benmansour R, Braun O, Hanafi S, Mladenovic N (2019) Using a variable neighborhood search to solve the single processor scheduling problem with time restrictions. Lect Notes Comput Sci 11328:202–215

Braun O, Chung F, Graham R (2014) Single-processor scheduling with time restrictions. J Sched 17:399–403

Braun O, Chung F, Graham R (2016) Worst-case analysis of the LPT algorithm for single processor scheduling with time restrictions. OR Spectrum 38:531–540

Kravchenko SA, Werner F (1997) Parallel machine scheduling problems with a single server. Math Comput Model 26:1–11

Rustogi K, Strusevich VA (2013) Parallel machine scheduling: impact of adding extra machines. Oper Res 61:1243–1257

Ruiz R, Vallada E, Fernández-Martínez C (2009) Scheduling in flowshops with no-idle machines. In: Uday KC (ed) Computational intelligence in flow shop and job shop scheduling. Springer, Berlin, pp 21–51

Zhang A, Chen Y, Chen L, Chen G (2017) On the NP-hardness of scheduling with time restrictions. Discret Optim 28:54–62

Zhang A, Ye F, Chen Y, Chen G (2017) Better permutations for the single-processor scheduling with time restrictions. Optim Lett 11:715–724