Supplementary Material (#7381)

Proof of Theorem 2

Proof. We reduce from UNARY BIN PACKING parameterized by the number k of bins, which is known to be W[1]-hard (Jansen et al. 2013). Given an instance I of UNARY BIN PACKING with k bins, each of size B and n objects $o_1, o_2, \dots o_n$ of size $s_1, s_2, \dots s_n$ such that $\sum_{i=1}^{n} s_i = kB$, we construct an instance I' of $1|nr = 1|C_{\max}$ as follows. We denote the set of all objects by O.

First, for each $o_i \in O$, we define a job $J_i = (p_i, a_i)$ such that $p_i = a_i = s_i$; we denote the set of all jobs by $\mathcal{J}$. Next, we construct k supply dates $q_i = (u_i, \tilde{b}_i)$, with $u_i = (i-1)B$ and $\tilde{b}_i = B$ for each $i \in [k]$. This way we obtain an instance I' of $1|nr = 1|C_{\max}$.

It remains to show that I is a yes-instance if and only if I' is an instance with $C_{\max} = kB$. To this end, suppose first that I is a yes-instance. Then, there is a partition of the objects into k bins such that each bin is (exactly) full. Formally, there are k sets $S_1, S_2, \dots S_k$ such that $\bigcup_i S_i = O$, $S_i \cap S_j = \emptyset$ for all $i \neq j$, and $\sum_{o_i \in S_j} s_i = B$ for all j. Hence we schedule all jobs j_i with $o_i \in S_1$ between time $0 = u_1$ and $B = u_2$. Following the same procedure, we schedule all jobs corresponding to objects in S_i between time u_i and u_{i+1} where $u_{i+1} = iB$. Since $\sum_{i=1}^{n} p_i = kB$, we conclude that $C_{\max} = kB$.

Now suppose that $C_{\max} = kB$. Assume towards a contradiction that there is an optimal schedule in which some job $J_i \in \mathcal{J}$ starts at some time t such that $t < u_\ell < t + p_i$ for some $\ell \in [k]$. Let S be the set of all objects that are scheduled before J_i. Since $C_{\max} = kB$, it follows that at each point in time until t, there is some job scheduled at this time. Thus, since $p_h = a_h$ for all jobs J_h, it follows that $\sum_{J_h \in S} a_h = \sum_{J_h \in S} p_h = t$. As a result, $\sum_{J_h \in S \cup \{J_i\}} a_h = t + a_i = t + p_i > u_\ell = (\ell - 1)B = \sum_{h=1}^{\ell-1} \tilde{b}_h$; a contradiction to j_i starting before u_ℓ. Hence, there is no job that starts before some u_ℓ and ends after it. Thus, the jobs can be partitioned into "phases," that is, there are k sets $T_1, T_2, \dots, T_k$ such that $\bigcup_h T_h = \mathcal{J}$, $T_i \cap T_{i'} = \emptyset$ for all $i \neq i'$, and $\sum_{J_h \in T_g} p_h = B$ for all g. This corresponds to a bin packing where o_g belongs to bin h if and only if $J_g \in T_h$. □

Proof of Lemma 1

Proof. Assuming that we have an oracle for $1, \text{NI}|nr = r|-$, we describe an algorithm solving $1|nr = r|C_{\max}$ that runs in polynomial time.

We first make a useful observation about feasible solutions to the original problem. Let us consider some feasible solution σ to $1|nr = r|C_{\max}$ and let $g_1, g_2, \dots, g_n$ be idle times before processing, respectively, jobs $J_1, J_2, \dots, J_n$. Then, σ can be transformed to another schedule σ' with the same makespan as σ and with idle times $g'_1, g'_2, \dots, g'_n$ such that $g'_2 = g'_3 \dots g'_n = 0$ and $g'_1 = \sum_{t \in [n]} g_t$. Intuitively, σ' is a scheduling in which the idle times of σ are all "moved" before the machine starts the first scheduled job. It is straightforward to see that in σ' no jobs are overlapping. Furthermore, each job according to σ' is processed at earliest at the same time as it is processed according to σ. Thus, because there are no "negative" supplies and the order of processed jobs is the same in both σ and σ', each job's resource request is met in scheduling σ'.

Using the above observation, the algorithm solving $1|nr = r|C_{\max}$ using an oracle for $1, \text{NI}|nr = r|-$ problem works as follows. First, it guesses a starting gap's duration $g \leq u_{\max}$ and then calls an oracle for $1, \text{NI}|nr = r|-$ subtracting g from each supply time (and merging all non-positive supply times to a new one arriving at time zero) of the original $1|nr = r|C_{\max}$ instance. For each value of g the algorithm adds g to the oracle's output and returns the minimum over all these sums.

Basically, the algorithm finds a scheduling with the smallest possible makespan assuming that the idle time happens only before the first scheduled job is processed. Thus, due to the initial observation that every schedule can be transferred to a same-makespan schedule that has only the initial idle time, the algorithm is correct. □

Proof of Lemma 2

Proof. Let σ be some feasible schedule. Consider a pair $(J_j, J_{j'})$ of jobs such that J_j is scheduled after $J_{j'}$ and J_j dominates $J_{j'}$, that is, $p_j \geq p_{j'}$, $a_{ij} \leq a_{ij'}$ for all $i \in \mathcal{R}$. Let us denote by σ' a schedule emerging from continuously scheduling all jobs in the same order as in σ but with jobs J_j and $J_{j'}$ swapped. Assume that σ' is not a feasible schedule. We show that each job in σ' meets its resource requirements thus contradicting the assumption and proving the lemma. We distinguish between the set of jobs $\mathcal{J}_{\text{out}}$ that are scheduled before $J_{j'}$ in σ or that are scheduled after J_j in σ and jobs $\mathcal{J}_{\text{in}}$ that are scheduled between j' and j in σ (including j' and j). Observe that since all jobs in $\mathcal{J}_{\text{in}}$ are scheduled without the machine idling, it holds that all jobs in $\mathcal{J}_{\text{out}}$ are scheduled exactly at the same times in both σ and σ'. Additionally, since the total number of resources consumed by jobs in $\mathcal{J}_{\text{in}}$ in both σ and σ' is the same, the resource requirements for each job in $\mathcal{J}_{\text{out}}$ is met in σ'. It remains to show that the requirements of all jobs in $\mathcal{J}_{\text{in}}$ are still met after swapping. To this end, observe that all jobs except for $J_{j'}$ still meet the requirements in σ' as J_j dominates $J_{j'}$ (i.e., J_j requires at most as many resources and has at least the same processing time as $J_{j'}$). Thus, each job in $\mathcal{J}_{\text{in}}$ except for $J_{j'}$ has at least as many resources available in σ' as they have in σ. Observe that $J_{j'}$ is scheduled later in σ' as J_j was scheduled in σ. Hence, there are also enough resources available in σ' to process $J_{j'}$. Thus, σ' is feasible, a contradiction. □

Proof of Theorem 3

Proof. We first show how to solve $1, \text{NI}|nr = 1,\ p_j = 1|-$. At the beginning, we order the jobs increasingly with respect to their requirements of the resource. Then, we simply check whether scheduling the jobs in the computed order yields a feasible schedule, that is, whether the resource requirement of each job is met. If the check fails, we return "no," otherwise we report "yes." The algorithm is correct due to Lemma 2 which, adapted to our case, says that there must exist an optimal schedule in which jobs with smaller

resource requirements are always processed before jobs with bigger requirements. It is straightforward to see that the presented algorithm runs in $O(n \log n)$ time.

To extend the algorithm to $1|nr = 1,\ p_j = 1|C_{\max}$, we apply Lemma 1. As described in detail in the proof of Lemma 1, we first guess the idling-time g of the machine at the beginning. Then, we run the algorithm for $1, \text{NI}|nr = 1,\ p_j = 1|-$ pretending that we start at time g by shifting backwards by g the times of all resource supplies. Since we can guess g using binary search in a range from 0 to the time of the last supply, such an adaptation adds a multiplicative factor of $O(\log n)$ to the running time of the algorithm for $1, \text{NI}|nr = 1,\ p_j = 1|-$. The correctness of the algorithm follows immediately from the proof of Lemma 1. □

Proof of Theorem 6

Proof. We will describe a dynamic-programming procedure that computes whether there exists a gapless schedule (Lemma 1). We next sort all of the jobs by their respective resource requirements using bucket sort in $O(n+a_{\max})$ time and then sort each of the buckets with respect to the processing times in $O(n \log n)$ time using merge sort. For the sake of simplicity, we will use $P_a[k]$ to describe the set of indices of the k jobs of resource requirement a and longest processing times. We next define the *ℓ^{th} phase* as the time interval from the start of any possible schedule up to $u_{\ell+1}$. Next, we define a dynamic program

$$T\colon [q-1]\times[n_1]\cup\{0\}\times\ldots\times[n_{a_{\max}}]\cup\{0\} \to \{true, false\},$$

where n_i is the number of jobs of resource requirement i. We want to store $true$ in $T[i, x_1, \ldots, x_{a_{\max}}]$ if and only if it is possible to schedule at least the jobs in $P_{a_k}[x_k]$ such that all of these jobs start within the i^{th} phase and there is no gap. If $T[q, n_1, \ldots, n_{a_{\max}}]$ is true, then this corresponds to a gapless schedule of all jobs and hence this is a solution. We will fill up the table T by increasing values of the first argument. That is, we will first compute all entries $T[1, x_1, x_2, \ldots, x_{a_{\max}}]$ for all possible combinations of values for $x_i \in [n_{a_i}]$. For the first phase, observe that $T[1, x_1, x_2, \ldots, x_{a_{\max}}]$ is set to true if and only if the two following conditions are met. First, $\sum_{i\in[a_{\max}]} x_k \cdot a_k \leq b_{11}$ as otherwise there are not enough resources available in the first i phases. Second, the sum of all processing times of all "selected jobs" without the longest one end at least one time steps before u_2 (as the job with the longest processing time can then be started at last in time step $u_2 - 1$ which is the last time step in the first phase). Formally, this can be written as $(\sum_{k\in[a_{\max}]} \sum_{j\in P_{a_k}[x_k]} p_j) - \max_{k\in[a_{\max}]}\{p_j \mid j \in P_{a_k}[1]\} \leq u_2 - 1$. For increasing values of the first argument, we do the following to compute $T[i, x_1, x_2, \ldots x_{a_{\max}}]$. We compute for all tuples of numbers $(y_1, y_2, \ldots y_{a_{\max}})$ with $y_k \leq x_k$ for all $k \in [a_{\max}]$ whether $T[i-1, y_1, y_2, \ldots, y_{a_{\max}}]$ is true and whether this schedule can be extended to a schedule for $T[i, x_1, x_2, \ldots, x_{a_{\max}}]$. This check needs to verify that there are enough resources for all selected jobs in the i^{th} phase and that there are no gaps in the schedule. The first check is very simple as we only need to check whether $\sum_{k\in[a_{\max}]} x_k \cdot a_k \leq b_{1i}$. To check that this schedule is gapless it is actually enough to check that $\sum_{k\in[a_{\max}]} \sum_{j\in P_{a_k}[y_k]} p_j \geq u_i - 1$ as if this was not the case but there still was a gapless schedule, then we could add some other job to the $i-1^{\text{th}}$ phase and since we iterate over all possible combinations of values of y_i, we will find this schedule in another iteration.

It remains to analyze the running time of this algorithm. First, there are $(q-1)\cdot\prod_{k\in[a_{\max}]}(n_k+1) \in O(q \cdot n^{a_{\max}})$ table entries. For each table entry $T[i, x_1, x_2, \ldots, x_{a_{\max}}]$, there are at most $\prod_{k\in[a_{\max}]}(x_k+1) \leq \prod_{k\in[a_{\max}]}(n_k+1) \in O(n^{a_{\max}})$ possible tuples $(y_1, y_2, \ldots, y_{a_{\max}})$ with $y_k \leq x_k$ for all $k \in [a_{\max}]$ and the two checks can be performed in $O(a_{\max})$ time. Thus, the overall running time is $O(q \cdot a_{\max} \cdot \log u_{\max} \cdot n^{2a_{\max}})$. □

Proof of Proposition 1

Proof. Given an instance I of UNARY BIN PACKING with k bins, each of size B and n objects $o_1, o_2, \ldots o_n$ of size $s_1, s_2, \ldots s_n$ such that $\sum_{i=1}^{n} s_i = kB$, we construct an instance I' of $1|nr = 2, p_j = 1|C_{\max}$ as follows.

For each $i \in [k]$ we add a supply $q_i = (u_i, \tilde{b}_{1,i}, \tilde{b}_{2,i}) := ((i-1)B, B, B(B-1))$; thus, we create k supply dates. For each object o_i for $i \in [n]$, we create an *object job* $j_i = (p_i, a_{1,i}, a_{2,i}) := (1, s_i, B - s_i)$. Additionally, we create $kB - n$ *dummy jobs*; each of them having processing time 1, no requirement of the resources of the first type, and requiring B resources of the second type. We refer to the constructed instance as I'.

We show that I is a yes-instance if and only if there is a schedule for the constructed instance I' with makespan exactly kB. Clearly, if I is a yes-instance, then there is a k-partition of the objects to subsets $S_1, S_2, \ldots, S_k$ such that, for each $i \in k$, $\sum_{o_j\in S_i} s_j = B$. For some set S_i, we schedule all jobs in S_i, one after another, starting at time $(i-1)B$. Then, in the second step, we schedule the remaining dummy jobs (if they exist) such that we obtain a gapless schedule σ. Naturally, since each S_i contains at most B objects, the object jobs are non-overlapping in σ. Since in the second step we have exactly $kB - n$ jobs available (recall that n is the number of object jobs) σ is a gapless schedules with makespan exactly kB. To check the feasibility of σ, let us consider the first B time steps. Note that from time 0 to time $|S_1|$, the machine processes all object jobs representing objects from S_1; from time $|S_1| + 1$ to B it processes exactly $B - |S_1|$ dummy jobs. Thus, the number of used resources of type 1 is $\sum_{o_j\in S_i} s_j = B$, and the number of used resources of type 2 is $\sum_{o_j\in S_i}(B - s_j) + (B - |S_1|)B = |S_1|B - B + B^2 - |S_1|B = B(B-1)$. As a result, at time $B-1$ there are no resources available, so we can apply the argument for the first B time steps to all following $k-1$ periods of B time steps; we eventually obtain that σ is a feasible schedule.

For the opposite direction, let σ be a schedule with makespan kB for instance I'. We again consider the first B time steps and a set J_{ph} of exactly B jobs processed in this time. Let A_1 and A_2 be the usage of the re-

source of, respectively, type 1 and type 2 by the jobs in J_{ph}. We denote by $J_{\text{o}} \subseteq J_{\text{ph}}$ the object jobs within J_{ph}. Then, $A_1 := \sum_{j \in J_{\text{o}}} a_{1,j} + \sum_{j \in J_{\text{ph}} \setminus J_{\text{o}}} a_{1,j}$. In fact, since each dummy job has no requirement of the resources of the first type, we have $A_1 = \sum_{j \in J_{\text{o}}} a_{1,j}$. Moreover, there are only B resources of type 1 available in the first B time steps, so it is clear that $A_1 \leq B$. Using the fact that, for each job j_i, it holds that $a_{2,i} = B - a_{1,i}$, we obtain the following:

$$\begin{aligned} A_2 &:= \sum_{j \in J_{\text{o}}} a_{2,j} + \sum_{j \in J_{\text{ph}} \setminus J_{\text{o}}} a_{2,j} \\ &= |J_{\text{o}}|B - A_1 + |J_{\text{ph}} \setminus J_{\text{o}}|B = B^2 - A_1. \end{aligned}$$

Using the above relation between A_1 and A_2, we show that $A_1 = B$. For the sake of contradiction, assume $A_1 < B$. Immediately, we obtain that $A_2 > B^2 - B = B(B-1)$, which is impossible since we only have $B(B-1)$ resources of the second type in the first B time steps of schedule σ. Thus, since $A_1 = B$, we use exactly B resources of the first type and exactly $(B-1)B$ resources of the second type in the first B time steps of σ. We can repeat the whole argument to all following $k-1$ periods of B time steps. Eventually, we obtain a solution to I by taking the objects corresponding to the object jobs scheduled in the subsequent periods of B-time steps.

The reduction is clearly applicable in polynomial time and the number of supply dates is a function of solely the number of bins in an instance of UNARY BIN PACKING. □

Proof of Proposition 2

Proof Sketch. The main observation needed to extend the ILP from Theorem 5 is that, by Lemma 2, given two jobs with the same resource requirement, there is always a schedule that first schedules the longer (dominating) jobs. In essence, for each phase and each possible resource requirement, a solution is still fully described by the respective number job jobs with that requirement scheduled in the phase (it is clear that among jobs with the same resource requirement we can schedule jobs with decreasing length).

For multiple resources, we describe the resource requirement of job j by a *resource vector* $\vec{s} = (a_{1,j}, a_{2,j}, \ldots, a_{r,j})$. We use the following non-negative integer variables:

1. $x_{w,\vec{s}}$ denoting the number of jobs with resource vector $\vec{s}$ being started in phase w,
2. $x^{\Sigma}_{w,\vec{s}}$ denoting the number of jobs with resource vector $\vec{s}$ being started between phase 1 to w,
3. $\alpha_{y,w}$ denoting the number of resources of type y available in the beginning of phase w,
4. d_w denoting the endpoint of phase w, that is, time point when the job started latest in phase w ends.

All constraints and proof arguments translate in a straightforward way. □

Proof of Theorem 7

Proof. We provide a parameterized reduction from the INDEPENDET SET problem which, given an undirected graph G and a positive integer k, asks whether there is an *independet set* of size k, that is, a set of k vertices in G which are pairwise non-adjacent. INDEPENDENT SET is W[1]-hard for the size k of the independet set.

Given an INDEPENDENT SET instance (G, k) we create an instance of $1|nr|C_{\max}$ as follows. Let $V(G) = \{v_1, \ldots, v_n\}$ and $E(G) = \{e_1, \ldots, e_m\}$. For each edge in G, we create one resource, that is, $nr = m$. At time point $u_1 = 0$, we provide one initial unit of every resource. At time point $u_2 = k$, we provide another unit of every resource. For each vertex $v_j \in V(G)$, there is one job J_j of length one with resource requirement being consistent with the incident edges, that is, $a_{i,j} = 1$ if $v_j \in e_i$ and $a_{i,j} = 0$ if $v_j \notin e_i$.

We claim that there is a schedule with makespan $C_{\max} = n$ if and only if there is an independent set of size k.

For the "if" direction, let $V' = \{v_{\ell_1}, \ldots, v_{\ell_k}\}$ be an independent set in G. Observe that every schedule that schedules the jobs $J_{\ell_1}, \ldots, J_{\ell_k}$ (in any order) in the first phase and then all other jobs (in any order) in the second phase is feasible and has makespan $C_{\max} = n$. Feasibility comes from the fact that we have an independent set so that no two jobs scheduled in the first phase require the same resource. (After the second supply, there are enough resources to schedule all jobs.) The makespan can be verified by observing that we have unit processing time and that the machine does not idle.

For the "only if" direction, assume that there is a feasible schedule with makespan $C_{\max} = n$ and let $\mathcal{J}' = \{J_{j_1}, \ldots, J_{j_k}\}$ be the jobs which are scheduled at time-points 0 to $k-1$. (Note that $\mathcal{J}'$ is well-defined since each job has length one, there are n jobs in total, and the makespan is n.) We claim that the set $V' = \{v_{j_1}, \ldots, v_{j_k}\}$ is an independent set. Assume towards a contradiction that two vertices are adjacent. Then, both jobs are scheduled in the first phase and require one unit of the the same edge resource; a contradiction. □