# Precedence-constrained scheduling problems parameterized by partial order width

René van Bevern[1,*], Robert Bredereck[2], Laurent Bulteau[3],
Christian Komusiewicz[4,**], Nimrod Talmon[5,***], and Gerhard J. Woeginger[6]

[1] Novosibirsk State University, Novosibirsk, Russian Federation, `rvb@nsu.ru`
[2] TU Berlin, Germany, `robert.bredereck@tu-berlin.de`
[3] Université Paris-Est Marne-la-Vallée, France, `l.bulteau@gmail.com`
[4] Friedrich-Schiller-Universität Jena, Germany, `christian.komusiewicz@uni-jena.de`
[5] Weizmann Institute of Science, Rehovot, Israel, `nimrodtalmon77@gmail.com`
[6] TU Eindhoven, The Netherlands, `gwoegi@win.tue.nl`

**Abstract.** Negatively answering a question posed by Mnich and Wiese (Math. Program. 154(1–2):533–562), we show that $P2|\text{prec}, p_j \in \{1, 2\}|C_{\max}$, the problem of finding a non-preemptive minimum-makespan schedule for precedence-constrained jobs of lengths 1 and 2 on two parallel identical machines, is W[2]-hard parameterized by the width of the partial order giving the precedence constraints. To this end, we show that SHUFFLE PRODUCT, the problem of deciding whether a given word can be obtained by interleaving the letters of $k$ other given words, is W[2]-hard parameterized by $k$, thus additionally answering a question posed by Rizzi and Vialette (CSR 2013). Finally, refining a geometric algorithm due to Servakh (Diskretn. Anal. Issled. Oper. 7(1):75–82), we show that the more general RESOURCE-CONSTRAINT PROJECT SCHEDULING problem is fixed-parameter tractable parameterized by the partial order width combined with the maximum allowed difference between the earliest possible and factual starting time of a job.

*Keywords:* resource-constrained project scheduling, parallel identical machines, makespan minimization, parameterized complexity, shuffle product

## 1 Introduction

We study the parameterized complexity of the following NP-hard problem and various special cases [13, 15] with respect to the width of the given partial order.

**Problem 1.1 (Resource-constrained project scheduling (RCPSP)).**
*Input:* A set $J$ of jobs, a partial order $\preceq$ on $J$, a set $R$ of renewable resources, for
   each resource $\rho \in R$ the available amount $R_\rho$, and for each $j \in J$ a processing
   time $p_j \in \mathbb{N}$ and the amount $r_{j\rho} \leq R_\rho$ of resource $\rho \in R$ that it consumes.

*Find:* A *schedule* $(s_j)_{j \in J}$, that is, a *starting time* $s_j \in \mathbb{N}$ of each job $j$, such that
1. for $i \prec j$, job $i$ finishes before job $j$ starts, that is, $s_i + p_i \leq s_j$,
2. at any time $t$, at most $R_\rho$ units of each resource $\rho$ are used, that is, $\sum_{j \in s(t)} r_{j\rho} \leq R_\rho$, where $s(t) := \{j \in J \mid t \in [s_j, s_j + p_j)\}$, and
3. the maximum completion time $C_{\max} := \max_{j \in J}(s_j + p_j)$ is minimum.

A schedule satisfying (1)–(2) is *feasible*; a schedule satisfying (1)–(3) is *optimal*.

Intuitively, a schedule $(s_j)_{j \in J}$ processes each job $j \in J$ non-preemptively in the half-open real-valued interval $[s_j, s_j + p_j)$, which costs $r_{j\rho}$ units of resource $\rho$ during that time. After finishing, jobs free their resources for later jobs. If there is only one resource and each job $j$ requires one unit of it, then RCPSP is equivalent to P|prec|$C_{\max}$, the NP-hard problem of non-preemptively scheduling precedence-constrained jobs on a given number $m$ of parallel identical machines to minimize the maximum completion time [15].

Mnich and Wiese [11] asked whether P|prec|$C_{\max}$ is solvable in $f(p_{\max}, w) \cdot \mathrm{poly}(n)$ time, where $p_{\max}$ is the maximum processing time, $w$ is the width of the given partial order $\preceq$, $n$ is the input size, and $f$ is a computable function independent of the input size. In other words, the question is whether P|prec|$C_{\max}$ is *fixed-parameter tractable* parameterized by $p_{\max}$ and $w$. Motivated by this question, which we answer negatively, we strengthen hardness results for P|prec|$C_{\max}$ and refine algorithms for RCPSP with small partial order width.

Due to space constraints, some details are deferred to a full version of this paper.

*Stronger hardness results.* We obtain new hardness results for the following special cases of P|prec|$C_{\max}$ (for basic definitions of parameterized complexity terminology, see the end of this section and recent textbooks [4, 5]):

(1) P2|chains|$C_{\max}$, the case with two machines and precedence constraints given by a disjoint union of total orders, remains weakly NP-hard for width 3.

(2) P2|prec,$p_j \in \{1, 2\}$|$C_{\max}$, the case with two machines and processing times 1 and 2, is W[2]-hard parameterized by the partial order width $w$.

(3) P3|prec,$p_j{=}1$,size$_j \in \{1, 2\}$|$C_{\max}$, the case with three machines, unit processing times, but where each job may require one or two machines, is also W[2]-hard parameterized by the partial order width $w$.

Towards showing (2) and (3), we show that SHUFFLE PRODUCT, the problem of deciding whether a given word can be obtained by interleaving the letters of $k$ other given words, is W[2]-hard parameterized by $k$. This answers a question of Rizzi and Vialette [12]. We put these results into context in the following.

Result (1) complements the fact that P|prec|$C_{\max}$ with constant width $w$ is solvable in pseudo-polynomial time using dynamic programming [14] and that P2|chains|$C_{\max}$ is *strongly* NP-hard for unbounded width [6].

Result (2) complements the NP-hardness result for P2|prec,$p_j \in \{1, 2\}$|$C_{\max}$ due to Ullman [15] and the W[2]-hardness result for P|prec,$p_j{=}1$|$C_{\max}$ parameterized by the number $m$ machines due to Bodlaender and Fellows [3]. While not made explicit, one can observe that Bodlaender and Fellows' reduction creates hard instances with $w = m + 1$. This is remarkable since P|prec|$C_{\max}$ is trivially polynomial-time solvable if $w \leq m$, and also since the result negatively

answered Mnich and Wiese's question [11] twenty years before it was posed. Our result (2), however, gives a stronger negative answer: unless $W[2] = FPT$, not even $P2|\text{prec},p_j \in \{1, 2\}|C_{\max}$ allows for the desired $f(w) \cdot \text{poly}(n)$-time algorithm.

*Refined algorithms.* Servakh [14] gave a geometric pseudo-polynomial-time algorithm for RCPSP with constant partial order width $w$. The degree of the polynomial depends on $w$ and, by (1) above, the algorithm cannot be turned into a true polynomial-time algorithm unless $P = NP$ even for constant $w$. We refine this algorithm to solve RCPSP in $(2\lambda+1)^w \cdot 2^w \cdot \text{poly}(n)$ time, where $\lambda$ is the maximum allowed difference between earliest possible and factual starting time of a job. The degree of the polynomial depends neither on $w$ nor $\lambda$ and is indeed a polynomial of the *input size $n$*. This does not contradict (1) since the factor $(2\lambda+1)^w$ might be superpolynomial in $n$. We note that fixed-parameter tractability for $w$ or $\lambda$ alone is ruled out by (2) and by Lenstra and Rinnooy Kan [10], respectively.

*Preliminaries.* A reflexive, symmetric, and transitive relation $\preceq$ on a set $X$ is a *partial order*. We write $x \prec y$ if $x \preceq y$ and $x \neq y$. A subset $X' \subseteq X$ is a *chain* if $\preceq$ is a total order on $X'$; it is an *antichain* if the elements of $X'$ are mutually incomparable by $\preceq$. The *width* of $\preceq$ is the size of largest antichain in $X$. A *chain decomposition* of $X$ is a partition $X = X_1 \uplus \cdots \uplus X_k$ such that each $X_i$ is a chain.

Recently, the *parameterized complexity* of scheduling problems attracted increased interest [2]. The idea is to accept exponential running times for solving NP-hard problems, but to restrict them to a small *parameter* [4, 5]. Instances $(x, k)$ of a *parameterized problem* $\Pi \subseteq \Sigma^* \times \mathbb{N}$ consist of an input $x$ and a parameter $k$. A parameterized problem $\Pi$ is *fixed-parameter tractable* if it is solvable in $f(k) \cdot \text{poly}(|x|)$ time for some computable function $f$. Note that the degree of the polynomial must not depend on $k$. FPT is the class of fixed-parameter tractable parameterized problems. There is a hierarchy of parameterized complexity classes $FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[P]$, where all inclusions are conjectured to be strict. A parameterized problem $\Pi_2$ is $W[t]$-*hard* if there is a *parameterized reduction* from each problem $\Pi_1 \in W[t]$ to $\Pi_2$, that is, an algorithm that maps an instance $(x, k)$ of $\Pi_1$ to an instance $(x', k')$ of $\Pi_2$ in time $f(k) \cdot \text{poly}(|x|)$ such that $k' \leq g(k)$ and $(x, k) \in \Pi_1 \Leftrightarrow (x', k') \in \Pi_2$, where $f$ and $g$ are arbitrary computable functions. No $W[t]$-hard problem is fixed-parameter tractable unless $FPT = W[t]$.

## 2 Parallel identical machines and shuffle products

This section presents our hardness results for special cases of $P|\text{prec}|C_{\max}$. In Section 2.1, we show weak NP-hardness of $P2|\text{chains}|C_{\max}$ for three chains. In Section 2.2, we show $W[2]$-hardness of SHUFFLE PRODUCT as a stepping stone towards showing $W[2]$-hardness of $P3|\text{prec},p_j{=}1,\text{size}_j \in \{1, 2\}|C_{\max}$ and $P2|\text{prec},p_j \in \{1, 2\}|C_{\max}$ parameterized by the partial order width in Section 2.3.

### 2.1 Weak NP-hardness for two machines and three chains

Du et al. [6] showed that $P2|\text{chains}|C_{\max}$ is strongly NP-hard. We complement this result by the following theorem.
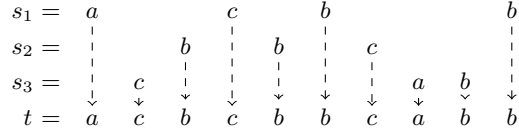
$$
\begin{array}{llllllllll}
s_1 = & a & & & c & & b & & & & b \\
s_2 = & & & b & & b & & c & & & \\
s_3 = & & c & & & & & & a & b & \\
t = & a & c & b & c & b & b & c & a & b & b
\end{array}
$$

**Fig. 2.1.** Illustration of a shuffle product: for $s_1 = acbb$, $s_2 = bbc$, and $s_3 = cab$, one has $t = acbcbbcabb \in s_1 \amalg s_2 \amalg s_3$. Dashed arcs show how the letters of each $s_i$ map into $t$.

**Theorem 2.1.** P2|chains|$C_{\max}$ is weakly NP-hard even for precedence constraints of width three, that is, consisting of three chains.

*Proof (sketch).* We reduce from the weakly NP-hard PARTITION problem [8, SP12]: Given a multiset of positive integers $A = \{a_1, \ldots, a_t\}$, decide whether there is a subset $A' \subseteq A$ such that $\sum_{a_i \in A'} a_i = \sum_{a_i \in A \setminus A'} a_i$. Let $A = \{a_1, \ldots, a_t\}$ be a PARTITION instance. If $b := \left(\sum_{a_i \in A} a_i\right)/2$ is not an integer, then we are facing a no-instance. Otherwise, we construct a P2|chains|$C_{\max}$ instance as follows. Create three chains $J^0 := \{j_1^0 \prec \cdots \prec j_t^0\}$, $J^1 := \{j_1^1 \prec \cdots \prec j_{t+1}^1\}$, and $J^2 := \{j_1^2 \prec \cdots \prec j_{t+1}^2\}$ of jobs. For each $i \in \{1, \ldots, t\}$, job $j_i^0$ gets processing time $a_i$. The jobs in $J^1 \cup J^2$ get processing time $2b$ each. This construction can be performed in polynomial time and one can show that the input PARTITION instance is a yes-instance of and only if the created P2|chains|$C_{\max}$ instance allows for a schedule with makespan $T := (2t + 3)b$: in such a schedule, each machine must perform exactly $t + 1$ jobs from $J^1 \cup J^2$ and has $b$ time for jobs from $J^0$. □

## 2.2 W[2]-hardness for Shuffle Product

In this section, we show a W[2]-hardness result for SHUFFLE PRODUCT that we transfer to P2|prec,$p_j \in \{1, 2\}$|$C_{\max}$ and P3|prec,$p_j=1$,size$_j \in \{1, 2\}$|$C_{\max}$ in Section 2.3. We first formally introduce the problem (cf. Figure 2.1).

**Definition 2.2 (shuffle product).** By $s[i]$, we denote the $i$th letter in a word $s$. A word $t$ is said to be in the *shuffle product* of words $s_1$ and $s_2$, denoted by $t \in s_1 \amalg s_2$, if $t$ can be obtained by interleaving the letters of $s_1$ and $s_2$. Formally, $t \in s_1 \amalg s_2$ if there are increasing functions $f_1 \colon \{1, \ldots, |s_1|\} \to \{1, \ldots, |t|\}$ and $f_2 \colon \{1, \ldots, |s_2|\} \to \{1, \ldots, |t|\}$ mapping positions of $s_1$ and $s_2$ to positions of $t$ such that, for all $i \in \{1, \ldots, |s_1|\}$ and $j \in \{1, \ldots, |s_2|\}$, one has $t[f_1(i)] = s_1[i]$, $t[f_2(j)] = s_2[j]$, and $f_1(i) \neq f_2(j)$. This product is associative and commutative, which implies that the shuffle product of any set of words is well-defined.

**Problem 2.3 ((Binary) Shuffle Product).**
*Input:* Words $s_1, \ldots, s_k$, and $t$ over a (binary) alphabet $\Sigma$.
*Parameter:* $k$.
*Question:* Is $t \in s_1 \amalg s_2 \amalg \cdots \amalg s_k$?

BINARY SHUFFLE PRODUCT is NP-hard for unbounded $k$ [16, Lemma 3.2], whereas SHUFFLE PRODUCT is polynomial-time solvable for constant $k$ using

dynamic programming. Rizzi and Vialette [12] asked about the parameterized complexity of Shuffle Product. We answer the question by the following theorem.

**Theorem 2.4.** Binary Shuffle Product is W[2]-hard.

Our proof uses a parameterized reduction from the W[2]-hard Dominating Set problem [4, 5] and is inspired by Bodlaender and Fellows's proof that $P|prec,p_j{=}1|C_{\max}$ is W[2]-hard parameterized by the number $m$ of machines [3].

**Problem 2.5 (Dominating Set).**
*Input:* A graph $G = (V, E)$ and a natural number $k$.
*Parameter:* $k$.
*Question:* Is there a size-$k$ *dominating set* $D$, that is, $V \subseteq N[D]$?

Herein, $N[D]$ is the set of vertices in $D$ and their neighbors. In order to describe the construction, we introduce some notation.

**Definition 2.6.** We denote the concatenation of words $s_1, \ldots, s_k$ as $\prod_{i=1}^{k} s_i := s_1 s_2 \ldots s_k$ and denote $k$ repetitions of a word $s$ by $s^k$. The number of occurrences of a letter $a$ in a word $s$ is $|s|_a$.

**Construction 2.7.** Given a Dominating Set instance $(G, k)$ with a graph $G = (V, E)$, we construct an instance of Binary Shuffle Product with $k + 3$ words over $\Sigma = \{a, b\}$ in polynomial time as follows. The construction is illustrated in Figure 2.2. Without loss of generality, assume that $V = \{1, \ldots, n\}$.

$$\text{For } u, v \in V, \text{ let } \ell_{u,v} := \begin{cases} 1 & \text{if } u = v \text{ or } \{u, v\} \in E, \\ 2 & \text{otherwise.} \end{cases} \tag{2.1}$$

Moreover, define two words

$$A := \prod_{u=1}^{n} \prod_{v=1}^{n} ab^{\ell_{u,v}} \qquad \text{and} \qquad B := \left((a^k b^{2k})^{n-1} a^k b^{2k-1}\right)^n.$$

Finally, let $N := 2k(n-1) + 1$ and output an instance of Shuffle Product with the following $k + 3$ words:

$$s_i := A^N \quad \text{for each } i \in \{1, \ldots, k\}, \qquad t := B^N (a^k b^{2k})^{n-1},$$

$$s_{k+1} := a^{|t|_a - \sum_{i=1}^{k} |s_i|_a}, \quad \text{and} \qquad s_{k+2} := b^{|t|_b - \sum_{i=1}^{k} |s_i|_b}.$$

Note that $A$ is simply the word that one obtains by concatenating the rows of the adjacency matrix of $G$ and replacing ones by $ab$ and zeroes by $abb$.

Before showing the correctness of Construction 2.7, we make some basic observations about the words it creates, for which we introduce some terminology.

**Definition 2.8 (long and short blocks, positions).** A *block* in a word $s$ is a maximal consecutive subword using only one letter. A *c-block* is a block containing only the letter $c$. A block has *position $i$* in $s$ if it is the $i$th successive block in $s$. We call $b$-blocks of length $2k - 1$ in $t$ *short* and $b$-blocks of length $2k$ *long*.
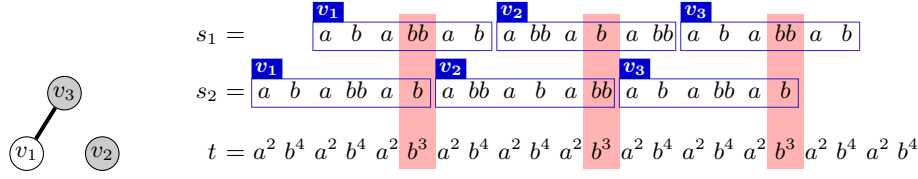
**Fig. 2.2.** Left: A DOMINATING SET instance with $k = 2$ and a solution $\{v_2, v_3\}$ (the gray nodes). Right: The "base pattern" of the corresponding SHUFFLE PRODUCT instance (only one repetition of $A$ in $s_1$ and $s_2$ and only one repetition of $B$ in $t$ is shown). Blocks of $s_1$ and $s_2$ are mapped into the blocks of $t$ displayed in the same column. The horizontal (blue) rectangles reflect that each $s_i$ is built as the concatenation of the rows of the adjacency matrix, where zeroes are replaced by *abb* and ones by *ab*. The amount of horizontal offset of each $s_i$ corresponds to the selection of a vertex as dominator ($v_2$ for $s_1$ and $v_3$ for $s_2$). The dark columns (red) correspond to the short *b*-blocks of $t$: they ensure that, in each row of the adjacency matrix, at least one selected vertex dominates the vertex corresponding to that row. The base pattern is repeated $N$ times to ensure that at least one occurrence of the pattern is mapped to $t$ without unwanted gaps. Additional words $s_{k+1}$ and $s_{k+2}$ are added to match the remaining letters from $t$.

**Observation 2.9.** The words $s_1, \ldots, s_k$ and $t$ created by Construction 2.7 from a DOMINATING SET instance $(G, k)$ have the following properties:

(i) Each $s_i$ for $i \in \{1, \ldots, k\}$ contains $2Nn^2$ blocks.
(ii) The word $t$ contains $2Nn^2 + 2(n - 1)$ blocks.
(iii) For $i \in \{1, \ldots, k\}$, all $a$-blocks in $s_i$ have length 1. All $a$-blocks of $t$ have length $k$.
(iv) For $h \in \{1, \ldots, Nn\}$, the $b$-blocks at position $2hn$ in $t$ are short. All other $b$-blocks in $t$ are long.
(v) For each $i \in \{1, \ldots, k\}$, $p \in \{0, \ldots, N - 1\}$, and $u, v \in \{1, \ldots, n\}$, the $b$-block at position $2pn^2 + 2n(u-1) + 2v$ in $s_i$ has length $\ell_{u,v}$: it corresponds to the entry in the $u$th row and $v$th column of the adjacency matrix of $G$.

Since Construction 2.7 runs in polynomial time and the number of words in the created SHUFFLE PRODUCT instance only depends on the size of the sought dominating set, for Theorem 2.4, it remains to prove the following lemma.

**Lemma 2.10.** *Let $s_1, \ldots, s_{k+2}$ and $t$ be the words created by Construction 2.7 from a DOMINATING SET instance $(G, k)$. Then $G$ has a dominating set of size $k$ if and only if $t \in s_1 \sqcup s_2 \sqcup \cdots \sqcup s_{k+2}$.*

*Proof.* ($\Rightarrow$) Assume first that $G = (V, E)$ has a dominating set $D = \{d_1, \ldots, d_k\}$. We describe $t$ as a shuffle product of the words $s_i$ as follows. For each $i \in \{1, \ldots, k\}$, map all letters from the block at position $x$ of $s_i$ into block $x + 2(n - d_i)$ of $t$, that is, consecutive blocks of $s_i$ are mapped into consecutive blocks of $t$ with a small offset depending on $d_i$. So far, at most $k$ letters are mapped into each $a$-block of $t$ and at most $2k$ letters are mapped into each $b$-block of $t$. Hence, all $a$-blocks and all long $b$-blocks of $t$ are long enough to accommodate all their

designated letters. It remains to show that at most $2k - 1$ letters are mapped into each short $b$-block $\beta$ of $t$. By Observation 2.9(iv), $\beta$ is at position $2hn$ for some $h \in \{1, \ldots, Nn\}$. Thus, there are $p \in \{0, \ldots, N - 1\}$ and $u \in \{1, \ldots, n\}$ such that $2hn = 2(pn + u)n = 2pn^2 + 2un$. For each $s_i$, the block $\alpha_i$ of $s_i$ mapped into $\beta$ has position $(2pn^2 + 2un) - 2(n - d_i) = 2pn^2 + 2(u - 1)n + 2d_i$. Hence, $\alpha_i$ has length $\ell_{u,d_i}$ by Observation 2.9(v). Since $D$ is a dominating set, it contains a vertex $d_{i^*}$ such that $d_{i^*} = u$ or $\{d_{i^*}, u\} \in E$. Thus, by (2.1), $\alpha_{i^*}$ has length $\ell_{u,d_{i^*}} = 1$. Overall, at most $k$ $b$-blocks of $\{s_1, \ldots, s_k\}$ are mapped into $\beta$. We have shown that at least one of them, namely $\alpha_{i^*}$, has length one. Since the others have length at most two, at most $2k - 1$ letters are mapped into block $\beta$.

We have seen a mapping of the words $s_i$ with $i \in \{1, \ldots, k\}$ to $t$. Thus, we have $|t|_a \geq \sum_{i=1}^{k} |s_i|_a$ and $|t|_b \geq \sum_{i=1}^{k} |s_i|_b$ and the words $s_{k+1}$ and $s_{k+2}$ are well-defined. It remains to map $s_{k+1}$ and $s_{k+2}$ to $t$. Since $s_{k+1}$ consists only of $a$ and $s_{k+2}$ only of $b$, we only have to check that $t$ contains as many letters $a$ and $b$ as all words $s_i$ together, which is true by the definition of $s_{k+1}$ and $s_{k+2}$. We conclude that $t \in s_1 \shuffle s_2 \shuffle \cdots \shuffle s_{k+2}$ if $G$ has a dominating set of size $k$.

($\Leftarrow$) Assume that $t \in s_1 \shuffle s_2 \shuffle \cdots \shuffle s_{k+2}$. We show that $G$ has a dominating set of size $k$. To this end, for $i \in \{1, \ldots, k\}$, let $y_i(x)$ be the position of the block in $t$ into which the last letter of the block at position $x$ of $s_i$ is mapped and let $\delta_i(x) = y_i(x) - x$. We will see that, intuitively, one can think of $\delta_i(x)$ as the shift of the $x$th block of $s_i$ in $t$. To show that $G$ has a dominating set of size $k$, we use the following two facts about $\delta_i$, which we will prove afterwards.

(i) For $i \in \{1, \ldots, k\}$ and $x \in \{1, \ldots, 2Nn^2\}$, one has $\delta_i(x) \in \{0, \ldots, 2(n-1)\}$.
(ii) There is a $p \in \{0, \ldots, N - 1\}$ such that, for all $i \in \{1, \ldots, k\}$, $\delta_i$ is constant over the interval $I_p = \{2pn^2 + 1, \ldots, 2(p+1)n^2 + 1\}$.

We now focus on a $p \in \{0, \ldots, N - 1\}$ as in (ii) and write $\delta_i$ for the value $\delta_i(x)$ taken for all $x \in I_p$. We show that $D := \{d_i = n - \delta_i/2 \mid k \in \{1, \ldots, k\}\}$ is a dominating set of size $k$ for $G$, that is, we show $D \subseteq V$ and $V \subseteq N[D]$.

To this end, consider a vertex $u \in V$ and the block $\beta$ of $t$ at position $2pn^2 + 2un = 2hn$ for $h = pn + u \in \{1, \ldots, Nn\}$. By Observation 2.9(iv), $\beta$ is a short $b$-block. For any $i \in \{1, \ldots, k\}$, let $\alpha_i$ be the block at position $2pn^2 + 2un - \delta_i$ in $s_i$. Because of (i), this position is in $I_p$. By definition of $\delta_i$, the last letter of $\alpha_i$ is mapped into $\beta$. Thus, $\alpha_i$ is a $b$-block. Note that this implies that $\delta_i$ is even since $a$-blocks and $b$-blocks are alternating in $t$ and $s_i$. Moreover, by (i), $d_i = n - \delta_i/2 \in \{1, \ldots, n\} = V$. It follows that $D \subseteq V$. We show that $u \in N[D]$. To this end, note that the $a$-block in $s_i$ at position $2pn^2 + 2un - \delta_i - 1 \in I_p$ directly preceding $\alpha_i$ is mapped into the $a$-block of $t$ at position $2pn^2 + 2un - 1$ directly preceding $\beta$. Thus, all letters of $\alpha_i$ are mapped into $\beta$ and one has

$$\sum_{i=1}^{k} |\alpha_i| \leq |\beta|. \tag{2.2}$$

By Observation 2.9(v), $\alpha_i$ has length $\ell_{u,(n-\delta_i/2)} = \ell_{u,d_i}$. Since $\beta$ is a short $b$-block, it has length $2k - 1$. From (2.2), we get $\sum_{i=1}^{k} \ell_{u,d_i} \leq 2k - 1$. Thus, there

is some $i^* \in \{1, \ldots, k\}$ with $\ell_{u,d_{i*}} = 1$. By (2.1), that means $d_{i*} = u$ or $\{u, d_{i*}\}$ is an edge in $G$. Hence, $u \in N[D]$ and $D$ is a dominating set of size $k$ for of $G$.

It remains to prove (i) and (ii). For (i), note that $y_i(1) \geq 1$ and $y_i(x+1) \geq y_i(x) + 1$. Hence, $\delta_i$ is non-decreasing with all values being non-negative. Furthermore, for $x = 2Nn^2$, $y_i(x) \leq 2Nn^2 + 2(n-1)$ since $t$ has only so many blocks by Observation 2.9(ii). Thus, the maximum possible value of $\delta_i$ is $2(n-1)$. Towards (ii), we say that a value of $p \in \{0, \ldots, N-1\}$ is *bad for i* if $\delta_i$ is not constant over $I_p$. For such a $p$, one has $\delta_i(2pn^2+1) < \delta_i(2(p+1)n^2+1)$. Hence, there can be at most $2(n-1)$ values of $p$ that are bad for $i$. Overall, there are at most $2k(n-1) < N$ values of $p$ that are bad for some $i \in \{1, \ldots, k\}$. Thus, at least one value is not bad for any $i$. For this value of $p$, every $\delta_i$ is constant over the interval $I_p$. $\square$

## 2.3   W[2]-hardness of scheduling problems parameterized by width

In the previous section, we showed W[2]-hardness of SHUFFLE PRODUCT. We now transfer this result to scheduling problems on parallel identical machines.

**Theorem 2.11.** The following two problems are W[2]-hard parameterized by the width of the partial order giving the precedence constraints.
   (i) P2|prec,$p_j \in \{1,2\}$|$C_{\max}$,     (ii) P3|prec,$p_j=1$,size$_j \in \{1,2\}$|$C_{\max}$.

We prove (i) using the following parameterized reduction from SHUFFLE PRODUCT with $k+1$ words to P2|prec,$p_j \in \{1,2\}$|$C_{\max}$ with $k+2$ chains.

**Construction 2.12.** Let $(s_1, \ldots, s_k, t)$ be a SHUFFLE PRODUCT instance over the alphabet $\Sigma = \{1, 2\}$. Assume that $|t|_1 = \sum_{i=1}^{k} |s_i|_1$ and $|t|_2 = \sum_{i=1}^{k} |s_i|_2$ (otherwise, it is a no-instance). We create an instance of P2|prec,$p_j \in \{1,2\}$|$C_{\max}$:
   (1) For each $i \in \{1, \ldots, k\}$, create a chain of *worker* jobs $j_{i1} \prec j_{i2} \prec \cdots \prec j_{i|s_i|}$, where $j_{i,x}$ has length $s_i[x]$.
   (2) For each $x \in \{1, \ldots, |t|\}$, create three *floor* jobs $z_{x,1}, z_{x,2}, z_{x,3}$ with $z_{x,1} \prec z_{x,2}$ and $z_{x,1} \prec z_{x,3}$, where $z_{x,1}$ has length $t[x]$, and $z_{x,2}$ and $z_{x,3}$ have length 1. If $x < |t|$, then also add the precedence constraints $z_{x,2} \prec z_{x+1,1}$ and $z_{x,3} \prec z_{x+1,1}$.

Observe that $\{z_{x,1}, z_{x,2} \mid 1 \leq x \leq |t|\}$ is chain. Thus, the makespan of any schedule is at least $T := \sum_{x=1}^{|t|} (t[x] + 1)$. For $x \in \{1, \ldots, n\}$, let $\tau(x) := \sum_{i=1}^{x-1} (t[x] + 1)$.

**Observation 2.13.** A schedule with makespan $T$ must schedule job $z_{x,1}$ at time $\tau(x)$, and jobs $z_{x,2}$ and $z_{x,3}$ at time $\tau(x) + t[x]$. Thus, for $x \in \{1, \ldots, |t|\}$, both machines are used by floor jobs from $\tau(x) + t[x]$ to $\tau(x) + t[x] + 1$ and one machine is free of floor jobs between $\tau(x)$ and $\tau(x) + t[x]$ for $t[x]$ time units. We call these *available time slots*.

Construction 2.12 runs in polynomial time. Moreover, from $k+1$ input words, it creates instances of width $k+2$: there are $k$ chains of worker jobs and the floor decomposes into two chains $\{z_{x,1}, z_{x,2} \mid 1 \leq x \leq |t|\}$ and $\{z_{x,1}, z_{x,3} \mid 1 \leq x \leq |t|\}$. To prove Theorem 2.11(i), one can thus show that $t \in s_1 \sqcup \cdots \sqcup s_k$ if and only if the created P2|prec,$p_j \in \{1,2\}$|$C_{\max}$ instance allows for a schedule of makespan $T$. By

Observation 2.13, any such schedule has available time slots of lengths corresponding to the letters in $t$, each of which can accommodate a worker job corresponding to a letter of $s_1, \ldots, s_k$. The precedence constraints ensure that these worker jobs get placed into the time slots corresponding to letters of $t$ in increasing order.

The proof of Theorem 2.11(ii) works analogously: one simply replaces worker jobs of length two by worker jobs of length one that require two machines and modifies the floor jobs so that they do not create time slots of length one or two, but so that each created time slot is available on only one or on two machines. To achieve this, the construction uses three machines.

# 3  Resource-Constrained Project Scheduling

In Section 2.3, we have seen that P3|prec,$p_j$=1,size$_j\in\{1,2\}|C_{\max}$ is W[2]-hard parameterized by the partial order width. It follows that also RCPSP (cf. Problem 1.1) is W[2]-hard for this parameter, even if the number of resources and the maximal resource usage are bounded by two and all jobs have unit processing times. In this section, we additionally consider the *lag* parameter:

**Definition 3.1 (earliest possible starting time, lag).** Let $J_0 \subseteq J$ be the jobs that are minimal elements in the partial order $\preceq$. The *earliest possible starting time* $\sigma_j$ is 0 for a job $j \in J_0$ and, inductively, $\max_{i \prec j}(\sigma_i + p_i)$ for a job $j \in J \setminus J_0$. The *lag* of a feasible schedule $(s_j)_{j \in J}$ is $\lambda := \max_{j \in J}(s_j - \sigma_j)$.

Lenstra and Rinnooy Kan's NP-hardness proof for P|prec,$p_j$=1$|C_{\max}$ [10] shows that it is even NP-hard to decide whether there is a schedule of makespan at most three and lag at most one. Thus, the lag $\lambda$ alone cannot lead to a fixed-parameter algorithm for RCPSP, just as the width $w$ alone cannot. We show a fixed-parameter algorithm for the parameter $\lambda + w$.

**Theorem 3.2.** An optimal schedule with lag at most $\lambda$ for RCPSP is computable in $(2\lambda + 1)^w \cdot 2^w \cdot \text{poly}(n)$ time if it exists, where $w$ is the partial order width.

Our algorithm is a refinement of Servakh's pseudo-polynomial-time algorithm for RCPSP with constant width [14], which is based on graphical optimization methods introduced by Akers [1] and Hardgrave and Nemhauser [9] for hand-optimizing JOB SHOP schedules for two jobs. We provide a concise translation of Servakh's algorithm in Section 3.1 before we prove Theorem 3.2 in Section 3.2.

## 3.1  Geometric interpretation of RCPSP

Given an RCPSP instance with precedence constraints $\preceq$ of width $w$, by Dilworth's theorem, we can decompose our set $J$ of jobs into $w$ pairwise disjoint chains. More specifically, these chains are efficiently computable [7]. For $\ell \in \{1, \ldots, w\}$, denote the jobs in chain $\ell$ by a sequence $(j_{\ell k})_{k=1}^{n_\ell}$ such that $j_{\ell k} \prec j_{\ell k+1}$ and let

$L_\ell^i := \sum_{k=1}^i p_{j_{\ell k}}$ be the sum of processing times of the first $i$ jobs on chain $\ell$,
$L_\ell := L_\ell^{n_\ell}$ be the sum of processing times of all jobs on chain $\ell$.

Let $\mathbf{0} := (0, \ldots, 0) \in \mathbb{R}^w$ and $\boldsymbol{L} := (L_1, \ldots, L_w)$. Each point in the $w$-dimensional orthotope $X := \{\boldsymbol{x} \in \mathbb{R}^w \mid \mathbf{0} \le \boldsymbol{x} \le \boldsymbol{L}\}$ describes a *state* as follows.

**Definition 3.3 (running, completed, feasibility).** Let $\boldsymbol{x} = (x_1, \ldots, x_w) \in X$. For each chain $\ell \in \{1, \ldots, w\}$, if $x_\ell \in [L_\ell^{i-1}, L_\ell^i)$, then the jobs $(j_{\ell k})_{k=1}^{i-1}$ of chain $\ell$ are *completed* and job $j_{\ell i}$ has been *processed for* $x_\ell - L_\ell^{i-1}$ *time*. We call job $j_{\ell i}$ *running* if $L_\ell^{i-1} < x_\ell < L_\ell^i$. We denote by

$J(\boldsymbol{x}) \subseteq J$ the set of jobs running in state $\boldsymbol{x}$ and by
$C(\boldsymbol{x}) \subseteq J$ the set of jobs completed in state $\boldsymbol{x}$.

A point $\boldsymbol{x} \in X$ is *feasible* if it holds that both

(F1) the jobs $J(\boldsymbol{x})$ comply with resource constraints, that is, $\sum_{j \in J(\boldsymbol{x})} r_{j\rho} \le R_\rho$ for each resource $\rho \in R$, and
(F2) if there are two jobs $i \prec j$ such that $j \in J(\boldsymbol{x})$, then $i \in C(\boldsymbol{x})$.

Note that points $\boldsymbol{x} \in X$ may indeed violate (F2): there are not only precedence constraints between jobs on one chain, but also between jobs on different chains.

Each feasible schedule now yields a path of feasible points in the orthotope $X$ from the point $\mathbf{0}$, where no job has started, to the point $\boldsymbol{L}$, where all jobs are completed. Each such path consists of (linear) segments of the form $[\boldsymbol{x}, \boldsymbol{x} + t\boldsymbol{\delta}]$ for some $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_w) \in \{0, 1\}^w$, which corresponds to running exactly the jobs on the chains $\ell$ with $\delta_\ell = 1$ for $t$ units of time. Since all processing times and starting times are integers (cf. Problem 1.1), we can assume $t \in \mathbb{N}$.

**Definition 3.4 (feasibility of segments and their lengths).** The *length* of a segment $[\boldsymbol{x}, \boldsymbol{x} + t\boldsymbol{\delta}]$ is $t$. The *length* of a path is the sum of the lengths of its segments. A segment $[\boldsymbol{x}, \boldsymbol{x} + t\boldsymbol{\delta}]$ is *feasible* if it contains only feasible points and interrupts no jobs; that is, if there is a job $j \in J(\boldsymbol{x})$ on chain $\ell$, then $\delta_\ell = 1$.

There is now a one-to-one correspondence between feasible schedules and paths from $\mathbf{0}$ to $\boldsymbol{L}$ consisting only of feasible segments and between the shortest of these paths and optimal schedules. This leads to the following algorithm.

**Algorithm 3.5 (Servakh [14]).** Compute a shortest feasible path from $\mathbf{0}$ to $\boldsymbol{L}$ using dynamic programming: for each feasible point $\boldsymbol{x} \in X \cap \mathbb{N}^w$ in lexicographically increasing order, compute the length $P(\boldsymbol{x})$ of a shortest feasible path from $\mathbf{0}$ to $\boldsymbol{x}$ using the recurrence relation

$$P(\mathbf{0}) = 0, \quad P(\boldsymbol{x}) = \min_{\boldsymbol{\delta} \in \Delta_{\boldsymbol{x}}} P(\boldsymbol{x} - \boldsymbol{\delta}) + 1 \quad \text{for feasible } \boldsymbol{x} \in X \cap \mathbb{N}^w \setminus \{\mathbf{0}\}, \quad (3.1)$$

where $\Delta_{\boldsymbol{x}}$ is the set of vectors $\boldsymbol{\delta} \in \{0, 1\}^w$ such that segment $[\boldsymbol{x} - \boldsymbol{\delta}, \boldsymbol{x}]$ is feasible.

To compute $P(\boldsymbol{L})$, one thus iterates over at most $\prod_{\ell=1}^w (L_\ell + 1)$ points $\boldsymbol{x} \in X \cap \mathbb{N}^w$, for each of them over $2^w$ vectors $\boldsymbol{\delta} \in \{0, 1\}^w$, and, for each, decides whether $[\boldsymbol{x} - \boldsymbol{\delta}, \boldsymbol{x}]$ is feasible. Since the set of running jobs is the same for all interior points of the segment, it is enough to check the feasibility of its end points and one interior point, which can be done in polynomial time. Thus, the algorithm runs in $\prod_{\ell=1}^w (L_\ell + 1) \cdot 2^w \cdot \mathrm{poly}(n)$ time, which is pseudo-polynomial for constant $w$.

## 3.2 Fixed-parameter algorithm for arbitrary processing times

The bottleneck of Algorithm 3.5 is that it searches for a shortest path from $\mathbf{0}$ to $\boldsymbol{L}$ in the whole orthotope $X$. For the case where we are only accepting schedules of maximum lag $\lambda$, we will shrink the search space significantly: we show that we only have to search for paths within a tight corridor around the path corresponding to the schedule $(\sigma_j)_{j \in J}$ that starts jobs at the earliest possible time.

**Definition 3.6 (point at time $t$ on a path).** Let $p$ be the path from $\mathbf{0}$ to $\boldsymbol{L}$ corresponding to a not necessarily feasible schedule $(s_j)_{j \in J}$ that, however, respects precedence constraints. Let $t \geq 0$ and $T$ be the length of $p$.

Then, $\boldsymbol{p}(t)$ is the endpoint of the subpath of length $t$ of $p$ starting in $\mathbf{0}$ for $t \leq T$, and $\boldsymbol{p}(t) := \boldsymbol{L}$ for $t > T$.

Since the definition requires $(s_j)_{j \in J}$ to respect precedence constraints, $\boldsymbol{p}(t)$ determines the state (cf. Definition 3.3) at time $t$ according to schedule $(s_j)_{j \in J}$.

**Definition 3.7 ($\lambda$-corridored).** Let $p$ be the path corresponding to the schedule $(\sigma_j)_{j \in J}$ that starts jobs at the earliest possible time (cf. Definition 3.1).

$$\Gamma_\lambda(t) := \{\boldsymbol{x} \in X \mid \boldsymbol{p}(t) - \boldsymbol{\lambda} \leq \boldsymbol{x} \leq \boldsymbol{p}(t)\}, \quad \text{where } \boldsymbol{\lambda} = (\lambda, \dots, \lambda) \in \mathbb{N}^w.$$

We call a path $q$ $\lambda$-*corridored* if $\boldsymbol{q}(t) \in \Gamma_\lambda(t)$ for all $t \geq 0$.

Note that points on the path $p$ in Definition 3.7 may violate Definition 3.3(F1), but not (F2). One can show the following relation between $\lambda$-corridored paths and schedules of lag $\lambda$.

**Lemma 3.8.** A feasible schedule $(s_j)_{j \in J}$ has lag at most $\lambda$ if and only if its corresponding path $q$ is $\lambda$-corridored.

Lemma 3.8 allows us to compute a shortest feasible path from $\mathbf{0}$ to $\boldsymbol{L}$ using only points in $\Gamma_\lambda(t)$ for some $t$. Herein, we will exploit the following condition for checking whether a path segment can be part of a $\lambda$-corridored path.

**Lemma 3.9.** Let $[\boldsymbol{x}, \boldsymbol{x} + t\boldsymbol{\delta}]$ for $\boldsymbol{\delta} \in \{0, 1\}^w$. If $\boldsymbol{x} \in \Gamma_\lambda(t_0)$ and $\boldsymbol{x} + t\boldsymbol{\delta} \in \Gamma_\lambda(t_0 + t)$ for some $t_0 \geq 0$, then $\boldsymbol{x} + \tau\boldsymbol{\delta} \in \Gamma_\lambda(t_0 + \tau)$ for all $0 \leq \tau \leq t$.

*Proof.* Let $p$ be the path corresponding to schedule $(\sigma_j)_{j \in J}$ as in Definition 3.7 and let $\boldsymbol{\delta} = (\delta_1, \dots, \delta_w) \in \{0, 1\}^w$. For any $\tau \in [0, t]$, consider

$$\boldsymbol{x}^\tau = (x_1^\tau, \dots, x_w^\tau) := \boldsymbol{x} + \tau\boldsymbol{\delta} \qquad \text{and} \qquad \boldsymbol{y}^\tau = (y_1^\tau, \dots, y_\ell^\tau) := \boldsymbol{p}(t_0 + \tau).$$

By the prerequisites of the lemma, we have $\boldsymbol{y}^0 - \boldsymbol{\lambda} \leq \boldsymbol{x}^0 \leq \boldsymbol{y}^0$ and $\boldsymbol{y}^t - \boldsymbol{\lambda} \leq \boldsymbol{x}^t \leq \boldsymbol{y}^t$. We show $\boldsymbol{y}^\tau - \boldsymbol{\lambda} \leq \boldsymbol{x}^\tau \leq \boldsymbol{y}^\tau$ for any $\tau \in [0, t]$.

We start with $\boldsymbol{x}^\tau \leq \boldsymbol{y}^\tau$. For the sake of contradiction, assume that there is some chain $\ell$ and a $\tau \in [0, t]$ such that $x_\ell^\tau > y_\ell^\tau$. Then, $x_\ell^\tau > y_\ell^\tau \geq y_\ell^0 \geq x_\ell^0$. It follows that $\delta_\ell = 1$, which contradicts $\boldsymbol{x}^t \leq \boldsymbol{y}^t$ because, then,

$$x_\ell^t = x_\ell^0 + t = x_\ell^0 + \tau + (t - \tau) = x_\ell^\tau + (t - \tau) > y_\ell^\tau + (t - \tau) \geq y_\ell^{\tau + (t - \tau)} = y_\ell^t.$$

Now, we show $\boldsymbol{y}^\tau - \boldsymbol{\lambda} \leq \boldsymbol{x}^\tau$. Consider some chain $\ell$. If $\delta_\ell = 1$, then we have $y_\ell^\tau - \lambda \leq y_\ell^0 + \tau - \lambda \leq x_\ell^0 + \tau = x_\ell^\tau$ and we are fine. If $\delta_\ell = 0$ and there is a $\tau \in [0, t]$ such that $y_\ell^\tau - \lambda > x_\ell^\tau$, then $y_\ell^t - \lambda \geq y_\ell^\tau - \lambda > x_\ell^\tau = x_\ell^t$, contradicting $\boldsymbol{y}^t - \boldsymbol{\lambda} \leq \boldsymbol{x}^t$. $\square$

We can now prove the following result by computing recurrence (3.1) for each of the $(\lambda+1)^w$ feasible points $\boldsymbol{x} \in \Gamma_\lambda(t) \cap \mathbb{Z}^w$ for all $t \in \{0, \ldots, L\}$.

**Proposition 3.10.** An optimal schedule of lag at most $\lambda$ for RCPSP if it exists is computable in $(\lambda+1)^w \cdot 2^w \cdot \mathrm{poly}(L)$ time, where $L$ is the sum of all processing times and $w$ is the partial order width.

However, note that this is a fixed-parameter algorithm only for polynomial processing times, which is why we skip the proof and go on towards proving Theorem 3.2— a fixed-parameter algorithm that works for arbitrarily large processing times. To this end, we prove that all maximal segments of a path corresponding to a schedule with lag at most $\lambda$ start and end in one of $2 \cdot |J|$ hypercubes with edge length $2\lambda+1$.

**Lemma 3.11.** Let $q$ be the path of a feasible schedule $(s_j)_{j \in S}$ of lag at most $\lambda$ and let $t_2 \le t_1 \le t_2 + \lambda$. Then, $\boldsymbol{q}(t_1) \in \Gamma_{2\lambda}(t_2 + \lambda)$ (cf. Definition 3.7).

*Proof.* Consider the schedule $(\sigma_j)_{j \in J}$ that starts each job at the earliest possible time and its path $p$. Our aim is to show

$$\boldsymbol{p}(t_2 + \lambda) - 2\boldsymbol{\lambda} \le \boldsymbol{q}(t_1) \le \boldsymbol{p}(t_2 + \lambda),$$

where $\boldsymbol{\lambda} = (\lambda, \ldots, \lambda) \in \mathbb{N}^w$. By Lemma 3.8, $q$ is $\lambda$-corridored. Thus,

$$\boldsymbol{p}(t_1) - \boldsymbol{\lambda} \le \boldsymbol{q}(t_1) \le \boldsymbol{p}(t_1) \quad \text{and} \quad \boldsymbol{p}(t_2 + \lambda) - \boldsymbol{\lambda} \le \boldsymbol{q}(t_2 + \lambda) \le \boldsymbol{p}(t_2 + \lambda).$$

From this, one easily gets $\boldsymbol{q}(t_1) \le \boldsymbol{p}(t_1) \le \boldsymbol{p}(t_2 + \lambda)$. Moreover, one has

$$\boldsymbol{p}(t_2 + \lambda) - 2\boldsymbol{\lambda} \le \boldsymbol{q}(t_2 + \lambda) - \boldsymbol{\lambda} \le \boldsymbol{q}(t_2) + \boldsymbol{\lambda} - \boldsymbol{\lambda} = \boldsymbol{q}(t_2) \le \boldsymbol{q}(t_1). \quad \square$$

**Lemma 3.12.** Let $q$ be the path of a feasible schedule $(s_j)_{j \in S}$ of lag at most $\lambda$ and let $[\boldsymbol{x}, \boldsymbol{x} + t\boldsymbol{\delta}]$ be a maximal segment of $q$ such that the set $J(\boldsymbol{x} + \tau\boldsymbol{\delta})$ of running jobs (cf. Definition 3.3) is the same for all $\tau \in (0, t)$. Then,

$$\{\boldsymbol{x}, \boldsymbol{x} + t\boldsymbol{\delta}\} \subseteq \Gamma := \bigcup_{j \in J} \Gamma_{2\lambda}(\sigma_j + \lambda) \cup \bigcup_{j \in J} \Gamma_{2\lambda}(\sigma_j + p_j + \lambda),$$

where $(\sigma_j)_{j \in J}$ is the schedule that starts each job at the earliest possible time.

*Proof.* Let $t_0$ be chosen arbitrarily such that $\boldsymbol{q}(t_0) \in \{\boldsymbol{x}, \boldsymbol{x} + t\boldsymbol{\delta}\}$. By maximality of the segment, some job $j \in J$ is starting or ending at time $t_0$, that is, $t_0 = s_j$ or $t_0 = s_j + p_j$. Then, $\{\boldsymbol{x}, \boldsymbol{x} + t\boldsymbol{\delta}\} \subseteq \Gamma_{2\lambda}(\sigma_j + \lambda) \cup \Gamma_{2\lambda}(\sigma_j + p_j + \lambda)$ follows from $\sigma_j \le s_j \le \sigma_j + \lambda$ and Lemma 3.11. $\square$

We are now ready to show a fixed-parameter algorithm for RCPSP parameterized by length and maximum lag. That is, we prove Theorem 3.2.

*Proof (of Theorem 3.2).* We compute the shortest feasible $\lambda$-corridored path from the state $\boldsymbol{0}$, were no job has started, to the state $\boldsymbol{L}$, where all jobs have been completed (cf. Lemma 3.8). We use dynamic programming similarly to Algorithm 3.5. By Lemma 3.12, it is enough to consider those paths whose segments start and end in $\Gamma$. Thus, for each $\boldsymbol{x} \in \Gamma \cap \mathbb{N}^w$ in lexicographically increasing order, we compute the length $P(\boldsymbol{x})$ of a shortest $\lambda$-corridored path from $\boldsymbol{0}$ to $\boldsymbol{x}$ with segments starting and ending in $\Gamma$. To this end, for an $\boldsymbol{x} \in \Gamma \cap \mathbb{N}^w$, let $\Delta_{\boldsymbol{x}}$ be the set of vectors $\boldsymbol{\delta} \in \{0, 1\}^w$ such that,

(i) there is a smallest integer $t_{\boldsymbol{\delta}} \geq 1$ such that $x - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta} \in \Gamma$ and such that
(ii) the segment $[\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta}, \boldsymbol{x}]$ is feasible.

Then, $P(\boldsymbol{0}) = 0$ and, for feasible $x \in \Gamma \cap \mathbb{N}^w \setminus \{\boldsymbol{0}\}$, one has

$$P(\boldsymbol{x}) = \min\{P(\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta}) + t_{\boldsymbol{\delta}} \mid \boldsymbol{\delta} \in \Delta_{\boldsymbol{x}} \text{ and } \boldsymbol{x} \in \Gamma_\lambda(P(\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta}) + t_{\boldsymbol{\delta}})\},$$

where $\min \emptyset = \infty$ and the last condition on $\boldsymbol{x}$ uses Lemma 3.9 to ensure that we are indeed computing the length $P(\boldsymbol{x})$ of a $\lambda$-*corridored* path (cf. Definition 3.7) to $\boldsymbol{x}$: by induction, we know that $P(\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta})$ is the length of a shortest $\lambda$-corridored path to $\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta}$, and thus $\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta} \in \Gamma_\lambda(P(\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta}))$.

We have to discuss how to check (i) and (ii). One can check (ii) in polynomial time since it is enough to check feasibility at the end points and one interior point of the segment since the set of jobs running at the interior points of $[\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta}, \boldsymbol{x}]$ does not change: otherwise, since jobs are started or finished only at integer times, there is a maximal subsegment $[\boldsymbol{x}, \boldsymbol{x} - t \cdot \boldsymbol{\delta}]$ with $t \leq t_{\boldsymbol{\delta}} - 1$ where the set of running jobs does not change. Then $\boldsymbol{x} - t \cdot \boldsymbol{\delta} \in \Gamma$ by Lemma 3.12, contradicting the minimality of $t_{\boldsymbol{\delta}}$.

Towards (i), we search for the minimum $t_{\boldsymbol{\delta}} \geq 1$ such that $\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta} \in \Gamma$. Consider the schedule $(\sigma_j)_{j \in J}$ that schedules each job at the earliest possible time (cf. Definition 3.1). It is computable in polynomial time. By Lemma 3.12, we search for the minimum $t_{\boldsymbol{\delta}} \geq 1$ such that $\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta} \in \Gamma_{2\lambda}(\sigma_j + \lambda)$ or $\boldsymbol{x} - t_{\boldsymbol{\delta}} \cdot \boldsymbol{\delta} \in \Gamma_{2\lambda}(\sigma_j + p_j + \lambda)$ for some job $j \in J$. That is, by Definition 3.7, for each job $j$, we find the minimum $t_j \geq 1$ that solves a system of linear inequalities of the form $\boldsymbol{y} - 2\boldsymbol{\lambda} \leq \boldsymbol{x} - t_j \cdot \boldsymbol{\delta} \leq \boldsymbol{y}$, where $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_w) \in \{0, 1\}^w$. Writing $\boldsymbol{y} = (y_1, \ldots, y_w)$ and $\boldsymbol{x} = (x_1, \ldots, x_w)$, either $t_j = \max(\{1\} \cup \{x_\ell - y_\ell \mid \delta_\ell = 1\})$ is the minimum such $t_j$ or there is no solution for job $j$. Note that $t_j$ is an integer since $\boldsymbol{x}$ and $\boldsymbol{y}$ are integer vectors. Thus, $t_{\boldsymbol{\delta}} = \min_{j \in J} t_j$ is computable in polynomial time.

We conclude that we process each $\boldsymbol{x} \in \Gamma \cap \mathbb{N}^w$ in $2^w \cdot \mathrm{poly}(n)$ time. Moreover, $\Gamma$ contains at most $2 \cdot |J| \cdot (2\lambda + 1)^w$ integer points since each job $j \in J$ contributes at most $(2\lambda + 1)^w$ points in $\Gamma_{2\lambda}(\sigma_j + \lambda)$ and at most $(2\lambda + 1)^w$ points in $\Gamma_{2\lambda}(\sigma_j + p_j + \lambda)$. A total running time of $(2\lambda + 1)^w \cdot 2^w \cdot \mathrm{poly}(n)$ follows. $\square$

## 4  Conclusion

Our algorithm for RCPSP shows, in particular, that P3|prec,$p_j$=1|$C_{\max}$ is fixed-parameter tractable parameterized by the partial order width $w$ and allowed lag $\lambda$. Since the NP-hardness of this problem is a long-standing open question [8, OPEN8], it would be surprising to show W[1]-hardness of this problem for *any* parameter: this would exclude polynomial-time solvability unless FPT = W[1]. Thus, it makes sense to search for a fixed-parameter algorithm for P3|prec,$p_j$=1|$C_{\max}$ parameterized by $w$, whereas we showed that already P2|prec,$p_j \in \{1, 2\}$|$C_{\max}$ and P3|prec,$p_j$=1,size$_j \in \{1, 2\}$|$C_{\max}$ are W[2]-hard parameterized by $w$.

## References

[1] Akers, Jr., S.B.: A graphical approach to production scheduling problems. Oper. Res. 4(2), 244–245 (1956)

[2] van Bevern, R.: Fixed-parameter algorithms in operations research: Opportunities and challenges. Parameterized Complexity News 12(1), 4–6 (2016)

[3] Bodlaender, H.L., Fellows, M.R.: W[2]-hardness of precedence constrained $k$-processor scheduling. Oper. Res. Lett. 18(2), 93–97 (1995)

[4] Cygan, M., Fomin, F.V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S.: Parameterized Algorithms. Springer (2015)

[5] Downey, R.G., Fellows, M.R.: Fundamentals of Parameterized Complexity. Springer (2013)

[6] Du, J., Leung, J.Y.T., Young, G.H.: Scheduling chain-structured tasks to minimize makespan and mean flow time. Inform. Comput. 92(2), 219–236 (1991)

[7] Felsner, S., Raghavan, V., Spinrad, J.: Recognition algorithms for orders of small width and graphs of small Dilworth number. Order 20(4), 351–364 (2003)

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

[9] Hardgrave, W.W., Nemhauser, G.L.: A geometric model and a graphical algorithm for a sequencing problem. Oper. Res. 11(6), 889–900 (1963)

[10] Lenstra, J.K., Rinnooy Kan, A.H.G.: Complexity of scheduling under precedence constraints. Operations Research 26(1), 22–35 (1978)

[11] Mnich, M., Wiese, A.: Scheduling and fixed-parameter tractability. Math. Program. 154(1-2), 533–562 (2015)

[12] Rizzi, R., Vialette, S.: On recognizing words that are squares for the shuffle product. In: Proc. 8th CSR. LNCS, vol. 7913, pp. 235–245. Springer (2013)

[13] Schwindt, C., Zimmermann, J. (eds.): Handbook on Project Management and Scheduling, vol. 1. Springer (2015)

[14] Servakh, V.V.: Effektivno razreshimy sluchaj zadachi kalendarnogo planirovaniya s vozobnovimymi resursami. Diskretn. Anal. Issled. Oper. 7(1), 75–82 (2000)

[15] Ullman, J.: NP-complete scheduling problems. J. Comput. Syst. Sci. 10(3), 384–393 (1975)

[16] Warmuth, M.K., Haussler, D.: On the complexity of iterated shuffle. J. Comput. Syst. Sci. 28(3), 345 – 358 (1984)