The Complexity Landscape of Resource-Constrained Scheduling

Submission #5923 (Full Version)

Abstract

The Resource-Constrained Project Scheduling Problem (RCPSP) and its extension via activity modes (MRCPSP) are well-established scheduling frameworks that have found numerous applications in a broad range of settings related to artificial intelligence. Unsurprisingly, the problem of finding a suitable schedule in these frameworks is known to be NP-complete—however, aside from a few results for special cases, we have lacked an in-depth and comprehensive understanding of the complexity of the problems from the viewpoint of natural restrictions of the considered instances in a general setting.

In the first part of our paper, we develop new algorithms and give hardness-proofs in order to obtain a detailed complexity map of (M)RCPSP that settles the complexity of all 1024 considered variants of the problem defined in terms of explicit restrictions of natural parameters of instances. In the second part, we turn to implicit structural restrictions defined in terms of interactions between projects and resources—there, we use the treewidth of suitable graph representations to push the frontiers of tractability towards instances that remained out of reach for algorithms based purely on explicit restrictions considered in the first part.

Introduction

The RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM (RCPSP) provides a generic and well-established framework for the formal description of scheduling problems. RCPSP has been the subject of extensive theoretical as well as empirical research in the context of Artificial Intelligence (Smith and Pyle 2004; Kuster, Jannach, and Friedrich 2007; Varakantham, Fu, and Lau 2016; Song 2017), Operations Research and Scheduling (van Bevern et al. 2016; Fu, Varakantham, and Lau 2010; Fu, Varakantham, and Lau 2016); see also the survey (Kolisch and Padman 2001) and book (Artigues, Demassey, and Neron 2008) dedicated to the topic. RCPSP falls within the wider framework of so-called *scheduling problems* which are classical and have been at the focus of a vast and diverse amount of works (Brucker and Knust 2011; Schwindt and Zimmermann 2015); indeed, many variants of the problem can be written in the well-known 3-field notation (Blazewicz, Lenstra, and Kan 1983).

On a high level, in scheduling problems one is given a set of *activities* that have to be processed in a given time frame while adhering to certain conditions. Solutions to scheduling problems are also called *schedules*. RCPSP represents the subclass of scheduling problems where the processing of activities requires the use of resources; these have certain *capacities* that limit how many activities can be processed concurrently, and activities have certain *resource requirements* and *durations* which describe what resources each activity needs to be assigned to and for how long. It is assumed that every activity needs all its required resources at the same time and for the whole time of its duration and typically it is also assumed that an activity cannot be interrupted (one also calls this *non-preemptiveness*). Now for RCPSP, a schedule consists of an assignment of the activities to certain *points in time* (simply modeled by natural numbers) such that the time it takes to process all activities satisfies a given *makespan* bound. Often one requires a schedule to assign certain activities in a given precedence order.

A prominent generalisation of RCPSP that has received considerable attention (Bofill et al. 2017; Barrios, Ballestín, and Valls 2011; Poppenborg and Knust 2016) is based on the addition of activity modes, capturing scenarios where it is possible to complete activities in multiple ways—each possibly requiring different amounts of time and resources. This gives rise to the MULTI-MODE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM (MRCPSP).

Contribution. It is known that RCPSP is NP-complete, and in fact remains NP-complete even when we consider only a single resource and when there are no precedence constraints (van Bevern et al. 2016; Garey and Johnson 1975). However, so far we have lacked a comprehensive understanding of the complexity of these fundamental scheduling problems under explicit and natural restrictions of considered instances; interestingly, already Blazewicz, Lenstra and Kan (1983) called for such a theoretical investigation in their seminal paper which formalized RCPSP: "The obvious research program would be to determine the borderline between easy and hard resource constrained scheduling problems." For example, is (M)RCPSP restricted to instances of constant makespan and number of resources NP-hard, or does the problem become polynomial-time solvable?

Our first contribution is a complete complexity map for (M)RCPSP which takes into account all combina-

tions of variants arising from the following explicit restrictions/attributes:

- Fixed upper-bound on number of activities (n), number of resources (m), maximum duration of an activity (t), maximum capacity of a resource (c), makespan ($C_{\max}$), and/or on the number of activities that can use each resource ($r_{\deg}$);
- No precedence constraints ($\neg P$);
- "Simple" instances, where each activity only uses a single resource (S) (see, e.g., the work of Damay et al. (2007));
- Whether we consider modes (MRCPSP) or not (RCPSP);
- All numbers are encoded in unary[1] (U).

With the exception of the modes attribute, we will adopt the convention of listing the attributes considered in a given fragment in angular brackets—for instance, MRCPSP$\langle c, r_{\deg}\rangle$ refers to instances of MRCPSP where each resource has bounded capacity, and each resource is only used by a bounded number of activities.

Since each of the above attributes can be viewed as an independent binary "switch", altogether this amounts to 2^{10} considered fragments of (M)RCPSP. Our first contribution is a complete classification of all of these problems in terms of classical complexity theory; we show that 736 fragments are polynomial-time solvable and that 288 are NP-hard. This is achieved by a collection of 3 new hardness proofs (in addition to 4 known NP-hard cases) and 6 polynomial-time algorithms, utilizing a range of diverse algorithmic techniques and approaches. An illustration of our complexity map is provided in Figure 1.

In the second part of our paper, we shift our focus from explicit restrictions on instances to implicit ones. More specifically, we ask whether one can exploit the structure of interactions between activities and/or resources to lift any of the obtained polynomial-time algorithms towards more general classes of instances. A natural way of capturing such structure is the graph-theoretic concept of *treewidth* (Robertson and Seymour 1983). As our second major contribution, we show that treewidth allows us to push the frontiers of tractability for MRCPSP by extending two of the algorithms obtained in the first part of the paper. Specifically, we consider the treewidth of two graph representations:

1. the *activity graph*, which represents activities as vertices and adds edges between activities which share a resource;
2. the *resource graph*, which represents resources as vertices and adds edges between resources which share an activity.

First, we use dynamic programming to obtain a polynomial-time algorithm for MRCPSP$\langle U\rangle$ when the treewidth of the activity graph is bounded—a result which extends the polynomial-time tractability of MRCPSP$\langle n\rangle$ to cover unary instances with many activities. Our second result is then a polynomial-time algorithm for MRCPSP$\langle \neg P, C_{\max}\rangle$ when the treewidth of the resource graph is bounded, this time lifting the polynomial-time tractability of MRCPSP$\langle m, \neg P, C_{\max}\rangle$ to well-structured instances with a large number of resources. In terms of technical contribution, while using dynamic programming in conjunction with treewidth is by now a well-established approach, its use in the MRCPSP setting (and especially on the resource graph) was technically involved and non-trivial—indeed, even solving the special case where the resource graph has bounded size (i.e., MRCPSP$\langle \neg P, C_{\max}\rangle$) is not immediate (see our Theorem 6).

Related Work. While the treewidth of instances has not been considered for RCPSP yet, the parameter has found numerous applications in prominent subfields and problems that are relevant for AI research, such as SAT (Gottlob, Scarcello, and Sideri 2002; Samer and Szeider 2009), ILP (Ganian and Ordyniak 2018; Ganian, Ordyniak, and Ramanujan 2017) and CSP (Freuder 1982; Samer and Szeider 2010). It is worth noting that instances of low treewidth may arise naturally in a variety of problems and settings—for example, the treewidth of control flow graphs arising from goto-free programs is known to be at most 6 (Thorup 1998).

RCPSP is known to be polynomial-time solvable when the *poset width* of the precedence constraints is bounded (van Bevern et al. 2016; Servakh 2000). This can be viewed as a different implicit restriction from treewidth: instead of restricting activity-resource interactions, it bounds the number of activities which can run in parallel.

Preliminaries

For an integer i, we use $[i]$ as shorthand for $\{1, \ldots, i\}$. The function argmin refers to the argument of the minimum. We assume that $\mathbb{N}$ is the set of non-negative integers.

Problem Definition. An instance $\mathcal{I}$ of MRCPSP is a tuple $\langle A, R, C, \mathcal{M}, T, \mathcal{Q}, P, C_{\max}\rangle$, of

- $A = \{a_1, \ldots, a_n\}$ a set of *activities*;
- $R = \{r_1, \ldots, r_{m'}, r_{m'+1}, \ldots, r_{m'+m''}\}$ a set of *resources*, where we distinguish between m' *renewable* $(r_1, \ldots, r_{m'})$ and m'' *non-renewable* $(r_{m'+1}, \ldots, r_{m'+m''})$ resources, and let $m = m' + m''$;
- $C : R \to \mathbb{N}$ a mapping from resources to *capacities*;
- $\mathcal{M} = \{M_1, \ldots, M_n\}$ a set of (pairwise disjoint) *activity mode* sets, and let $B = \bigcup_{i\in[n]} M_i$ be the set of all modes;
- $T : B \to \mathbb{N} \setminus \{0\}$ a mapping from modes to *durations*;
- $\mathcal{Q} : B \to \mathbb{N}^m$ a mapping of modes to *resource requirements*;
- $<_P$ a strict partial order on A which represents *precedence constraints*;
- $C_{\max} \in \mathbb{N}$ is the allowed *makespan*; we also refer to numbers in $[C_{\max}] \cup \{0\}$ as *time points* or *time steps*.

A solution or *schedule* for $\mathcal{I}$ is a pair (ω, α), where ω is a mapping from each activity a_i to a mode $w_i \in M_i$ and α is a mapping from each a_i to a *starting time* in $[C_{\max}] \cup \{0\}$, satisfying the following four types of constraints.

Makespan constraints: For each activity a_i: $\alpha(a_i) + T(w_i) \leq C_{\max}$.

Resource constraints: For each resource r_ℓ:

- if r_ℓ is renewable, i.e., $\ell \in [m']$, for each time point $j \in [C_{\max}]$: $R_j[\ell] \leq R[\ell]$, where R_j denotes the vector of resource capacities being used at time point j—formally, $R_j = \sum_{i:\alpha(a_i)\leq j\leq\alpha(a_i)+T(w_i)} \mathcal{Q}(w_i)$; and

[1] This captures the distinction between weak and strong NP-hardness. Unary instances arise when encoding certain problems.

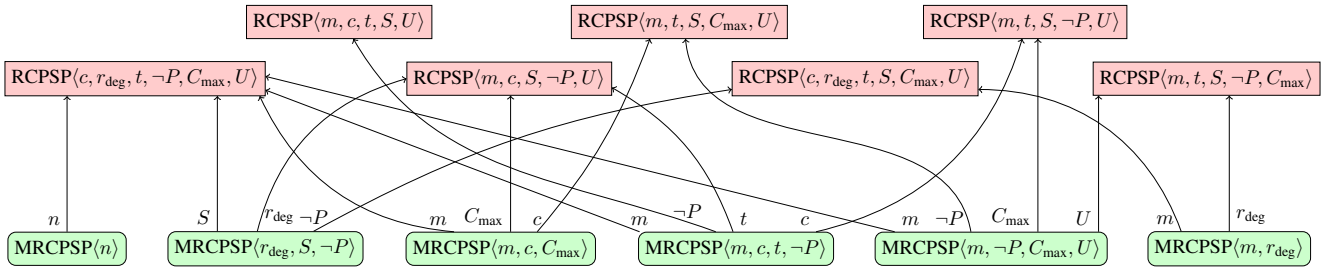

Figure 1: The obtained complexity map for MRCPSP showing polynomial-time solvable and NP-hard fragments of MRCPSP. Arrows indicate the tightness of the polynomially tractable fragments in terms of the flags, as removing any flag from a tractable fragment results in an NP-hard fragment.

– if r_ℓ is non-renewable, i.e., $\ell \in [m] \setminus [m']$, $\sum_{a_i \in A} \mathcal{Q}(w_i)[\ell] \leq R[\ell]$.

Precedence constraints: For each $a_i, a_{i'} \in A$ such that $w_i <_P w_{i'}$: $\alpha(a_i) + T(w_i) \leq \alpha(a_{i'})$.

The task in MRCPSP is to decide whether the instance admits a solution (in which case we also wish to compute such a solution), or not.

Following generic terminology in situations modeled by MRCPSP we also say, activity a_i is *processed* in a solution at the time steps in $\{\alpha(a_i), \ldots, \alpha(a_i) + T(w_i)\}$, and resource r_ℓ is being *used* or is *required* by a mode b if $\mathcal{Q}(b)[\ell] > 0$ and by a_i in a solution if w_i uses b in that solution.

RCPSP is the restriction of MRCPSP to the case where each activity has a single mode and all resources are renewable. In this case, we can simplify the notation by omitting $\mathcal{M}$ and having T and $\mathcal{Q}$ directly refer to activities in A (instead of modes).

The problem definition suggests a number of interesting and natural parameters which we want to consider as flags used to define the basic fragments of (M)RCPSP considered in this paper. A class $\mathcal{D}$ of MRCPSP instances has the flag $\langle n \rangle$ if there exists some integer z such that each instance in $\mathcal{D}$ has at most z activities. The flags $\langle m \rangle$ (total number of resources—renewable as well as non-renewable), $\langle t \rangle$ (maximum value of T), $\langle c \rangle$ (maximum value of C), $\langle C_{\max} \rangle$ are defined analogously. $\mathcal{D}$ has the flag $\langle r_{\deg} \rangle$ if there exists some integer z such that, for each instance $\mathcal{I} \in \mathcal{D}$ and for each resource r_ℓ in that instance, there are at most z activities which can use r_ℓ—formally, $|\{\, a_i \mid \forall b \in M_i, C(b)[\ell] = 0 \,\}| \geq n - z$. Intuitively, $\langle r_{\deg} \rangle$ represents a natural generalization of the flag $\langle n \rangle$, since it does not restrict the number of activities globally but only relatively to each resource.

Three of the four remaining flags—notably the ones signifying the lack of precedence constraints ($\langle \neg P \rangle$), an unary encoding of the numbers ($\langle U \rangle$)[2], and whether we have modes or not—are self-explanatory. The last remaining flag is $\langle S \rangle$ (short for "simple"), which signifies that for every activity a_i in an instance in the class $\mathcal{D}$, there is at most one resource used by a_i in any mode—formally, $\forall i \in [n]\, |\{\ell \in [m] \mid \exists b \in M_i\ \mathcal{Q}(b)[\ell] > 0\}| = 1$. Simple instances represent a middle ground between instances with a single resource and general instances and have a natural correspondence to classical scheduling over m types of machines (Gehrke et al. 2018), see also the work of Damay et al. (Damay, Quilliot, and Sanlaville 2007).

When our fragment has the U flag, we will use $|\mathcal{I}|$ to denote the size of a unary encoding of the instance $\mathcal{I}$, and otherwise $|\mathcal{I}|$ will be the size of a binary encoding of $\mathcal{I}$. In line with the 3-field notation for machine scheduling problems, we write the flags in the order $m, c, r_{\deg}, n, t, S, \neg, P, C_{\max}, U$; the first three flags imply restrictions on resources, the next four restrictions on activities and the second to last restricts the solution value. The additional flag U restricts the numbers one can encode within the size of the instance.

Treewidth and Graph Representations. Treewidth is a structural parameter which captures how "tree-like" a graph is. It has fundamental connections to graph theory and algorithms, and has been studied extensively in various fields. We refer to, e.g., the survey by Marx (2010) for an in-depth discussion of treewidth and its algorithmic applications.

A *tree-decomposition* $\mathcal{T}$ of a graph $G = (V, E)$ is a pair $(T, \mathcal{X})$, where T is a tree and $\mathcal{X}$ is a function that assigns each tree node t a set $\mathcal{X}(t) = X_t \subseteq V$ of vertices such that the following conditions hold:

- For every vertex $u \in V$, there is a tree node t such that $u \in X_t$.
- For every edge $uv \in E(G)$ there is a tree node t such that $u, v \in X_t$.
- For every vertex $v \in V(G)$, the set of tree nodes t with $v \in X_t$ forms a subtree of T.

The sets X_t are called *bags* of the decomposition $\mathcal{T}$ and X_t is the bag associated with the tree node t. The *width* of a tree-decomposition $(T, \mathcal{X})$ is the size of a largest bag minus 1. The *treewidth* of a graph G, denoted by $\mathrm{tw}(G)$, is the minimum width of a tree-decomposition of G.

Fact 1 (Bodlaender et al., 2016)**.** *There exists an algorithm which, given an n-vertex graph G and an integer k, in time $2^{\mathcal{O}(k)} \cdot n$ either outputs a tree-decomposition of G of width at most $5k+4$ and $\mathcal{O}(n)$ nodes, or determines that* $\mathrm{tw}(G) > k$.

Our algorithms will use a well-established canonical form of tree-decompositions. A tree-decomposition $\mathcal{T} = (T, \mathcal{X})$ is *nice* if T contains a root r (introducing natural ancestor-descendant relations in T) and the following holds:

- $|X_r| = |X_\ell| = 1$ for every leaf ℓ of T.

[2]We remark that establishing NP-hardness for a problem variant with the flag $\langle U \rangle$ is equivalent to showing *strong* NP-hardness.

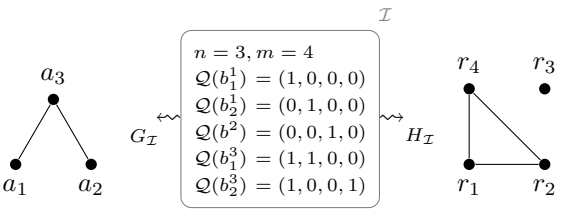

Figure 2: Graph representations of a MRCPSP-instance. (A mode is in M_i if i is its upper index, e.g., $b_2^1 \in M_1$.)

- There are only three kinds of non-leaf nodes in T:

  Introduce node: a note t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some vertex $v \notin X_{t'}$; we say that v is *introduced* at t. If $u \in X_{t'}$ and uv is an edge in G, then we also say that uv is *introduced* at t.

  Forget node: a note t with exactly one child t' such that $X_t = X_{t'} \setminus \{w\}$ for some vertex $w \in X_{t'}$; we say that v is *forgotten* at t.

  Join node: a node t with two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$.

There exists a polynomial-time algorithm that converts an arbitrary tree-decomposition into a nice tree-decomposition of the same width (Kloks 1994). We use $\mathcal{X}^{\downarrow}(t)$ to denote the set of all vertices in bags of the subtree of T rooted at t.

Here, we will consider two natural graph representations of a MRCPSP instance $\mathcal{I}$. The *activity graph* $G_\mathcal{I}$ has vertex set A and edges connect activities which may use the same resource, i.e., its edge set is $\{\, a_i a_j \mid \exists b \in M_i, b' \in M_j, \ell \in [m] \quad \mathcal{Q}(b)[\ell] \neq 0 \wedge \mathcal{Q}(b')[\ell] \neq 0 \,\}$. The *resource graph* $H_\mathcal{I}$ has vertex set R and edges connect resources which may be used by a single activity, i.e., its edge set is $\{\, r_i r_j \mid \exists \ell \in [n], b, b' \in M_\ell \quad \mathcal{Q}(b)[i] \neq 0 \wedge \mathcal{Q}(b')[j] \neq 0 \,\}$. Both graphs are depicted in Figure 2.

A Complexity Map for (M)RCPSP

In this section we give the polynomial-time algorithms and lower bounds (NP-hardness proofs) from which the complexity of all fragments obtained by considering any combination of considered flags follows (see Figure 1).

Polynomially Tractable Fragments

We present our six tractability results in an order roughly corresponding to the technical difficulty of the algorithms. Our first result is a simple observation identifying a basic polynomial-time fragment of MRCPSP.

Observation 2. MRCPSP$\langle n \rangle$ is in P.

Proof. MRCPSP$\langle n \rangle$ can be solved by the following algorithm $\mathcal{A}$. Given an instance $\mathcal{I}$ of the problem, $\mathcal{A}$ branches over all permutations of the activities and all assignments of activities to their modes. For such a choice of a permutation of the activities and an assignment of the activities to modes for each activity, whenever there is a solution (ω, α) for which ω is the chosen assignment of the activities to modes and α schedules activities in the order prescribed by the permutation, $\mathcal{A}$ constructs such a solution greedily: Define α for the activities in the order given by the permutation by setting $\alpha(a_i)$ to be smallest possible under the conditions that

- the makespan-, resource- and precedence constraints are not violated for the values of α that are already fixed, i.e., the values of α for activities occurring before a_i in the permutation under the current choice of ω; and
- $\alpha(a_i)$ is at least as large as all these values

whenever this is possible, and abandoning the current branch otherwise. If $\mathcal{A}$ finds a branch which results in a solution $\mathcal{A}$ outputs this solution, and returns that no solution exists otherwise. Correctness of $\mathcal{A}$ can be shown by observing that whenever a solution (ω, α) exists, there will be a branch where ω is the mapping of activities to modes and α orders activities in the same order as the considered permutation—and in such a branch, the greedy procedure is guaranteed to find either (ω, α) or a solution which differs from it by starting some activities sooner than α. All in all, $\mathcal{A}$ checks at most $n! \cdot \max_{i \in [n]} |M_i|^n$ branches, each of which requires at most quadratic time for processing. □

The following fragment—consisting of simple instances without precedence constraints and with a bound on the number of activities that use any particular resource—can be solved via a reduction to the MRCPSP$\langle n \rangle$ fragment.

Corollary 3. MRCPSP$\langle r_{\text{deg}}, S, \neg P \rangle$ is in P.

Proof. Since every activity uses at most a single resource (regardless of the mode it is set to), and since there are no precedence constraints between activities, an instance $\mathcal{I}$ of MRCPSP$\langle r_{\text{deg}}, S, \neg P \rangle$ can be split into a set of independent instances, each containing a single resource from $\mathcal{I}$. In particular, for $i \in [m]$ we define $\mathcal{I}_i$ to be the instance obtained from $\mathcal{I}$ by removing all resources other than r_i and all activities which do not use r_i; clearly, each such $\mathcal{I}_i$ has at most r_{deg} activities and hence can be solved in polynomial time by Observation 2. We let $\mathcal{I}_0$ be the instance containing all activities which do not use any resource at all (these can all be scheduled at time point 0, making $\mathcal{I}_0$ trivial). It is easy to see that if there is an $i \in [m] \cup \{0\}$ such that $\mathcal{I}_i$ is a NO-instance, then so is $\mathcal{I}$, and on the other hand a solution for $\mathcal{I}$ can be constructed as a union of the solutions for all $\mathcal{I}_i$, $i \in [m]$. □

We now proceed to fragments with non-trivial algorithms.

Theorem 4. MRCPSP$\langle m, c, C_{\max} \rangle$ is in P.

Proof. Any solution (ω, α) to an instance $\mathcal{I}$ contains at most $q = C_{\max} \cdot c \cdot m$ modes which use at least one resource—i.e., the set $B_{>0} = \{\, b \in \omega(A) \mid \exists j \in \mathbb{N} : \mathcal{Q}(b)[j] > 0 \,\}$ has cardinality at most q. Let $A_{>0} = \{a \in A \mid \omega(a) \in B_{>0}\}$ be the set of activities using these modes in the solution.

We can solve $\mathcal{I}$ using the algorithm $\mathcal{A}$ which begins by branching over all subsets of modes of cardinality at most q containing at most one mode from each of the pairwise disjoint M_i as options for $B_{>0}$. From such a choice for a possible $B_{>0}$ we infer a corresponding ω by setting $\omega(a_i) = b$ for $a_i \in A_{>0}$ whenever $B_{>0} \cap M_i = \{b\}$, and for all other activities $a_i \in A \setminus A_{>0}$ choosing $\omega(a_i)$ as $b \in M_i$ such that $\mathcal{Q}(b) = 0^m$ (i.e., b requires no resources) and minimizes $T(b)$ among these modes. It is easy to see that, whenever

a solution with the chosen $B_{>0}$ exists, a solution with the chosen $B_{>0}$ and deduced ω exists.

Now, we proceed similarly as in the proof of Observation 1 in which we branched on the order in which the activities are scheduled in a solution and then greedily constructed α which conforms to this ordering whenever such an α exists. The caveat here is that this exact approach would introduce a linear dependency on $n!$ which is in general not in $\mathrm{poly}(|\mathcal{I}|)$. Instead, $\mathcal{A}$ branches only on the order in which the activities in $A_{>0}$ are scheduled by a solution, inserts the activities in $A \setminus A_{>0}$ into this ordering, at the respective smallest positions respecting the precedence relation, and then a greedy starting time assignment is performed just as before. The overall runtime of $\mathcal{A}$ can be shown to lie in $\mathcal{O}(|B|^{C_{\max} \cdot c \cdot m} \cdot (C_{\max} \cdot c \cdot m)! \cdot |\mathcal{I}|^2) \subseteq \mathcal{O}((C_{\max} \cdot c \cdot m)! \cdot |\mathcal{I}|^{C_{\max} \cdot c \cdot m + 2})$. □

The proof strategy for Theorem 4 can be combined with that of Observation 2 to obtain a polynomial-time algorithm when $r_{\deg}$ and m are bounded. The resulting algorithm runs in time $\mathcal{O}((r_{\deg} \cdot m)! \cdot |\mathcal{I}|^{r_{\deg} \cdot m + 2})$.

Corollary 5. MRCPSP$\langle m, r_{\deg} \rangle$ is in P.

Proof. The proof is similar to that of Theorem 4. More concretely, one can begin by observing that in any solution there are at most $r_{\deg} \cdot m$ activities whose assigned modes use at least one resource, and proceed analogously from there on. □

The final two (and arguably most difficult) fragments for which we show polynomial-time tractability both have no precedence constraints and have boundedly many resources.

Theorem 6. MRCPSP$\langle m, \neg P, C_{\max}, U \rangle$ is in P.

Proof. Let a *resource snapshot* be a $C_{\max} \times m$ matrix over $[c] \cup \{0\}$ (i.e., the maximum capacity of a resource), and observe that the number of resource snapshots is upper-bounded by $(c+1)^{C_{\max} \cdot m}$. The resource snapshot J of a partial schedule (i.e., a solution restricted to a subset of activities) (ω, α) is the matrix where, for each $x \in [C_{\max}]$ and $y \in [m]$, the entry $J[x, y]$ is equal to the amount of resource r_y left at time step x—formally:

- if $y \leq m'$ then $J[x,y] = C(r_y) - (\sum_{a \in A', x - T(\omega(a)) \leq \alpha(a) \leq x} \mathcal{Q}(\omega(a))[y])$, and
- if $y > m'$ then $J[x,y] = C(r_y) - (\sum_{a \in A', \alpha(a) \leq x} \mathcal{Q}(\omega(a))[y])$.

Given an instance $\mathcal{I}$, let $\mathcal{J}_i$ be the set of resource snapshots of all partial schedule for the activities $\{ a_j \mid j < i \}$. Clearly, $\mathcal{J}_0$ contains a single resource snapshot, namely the one where $J[x, y] = C(r_y)$ for all x, y. On the other hand, if $\mathcal{J}_{n+1} \neq \emptyset$ then $\mathcal{I}$ is clearly a YES-instance.

To prove the theorem, we describe a dynamic programming algorithm $\mathcal{A}$ which computes $\mathcal{J}_{i+1}$ from $\mathcal{J}_i$. $\mathcal{A}$ begins by looping over all resource snapshots in $\mathcal{J}_i$, branching over each mode $b \in M_{i+1}$ of activity a_{i+1} and branching over each starting time $s \in [C_{\max} - T(b)]$. For each such choice of resource snapshot J, b and s, it creates a new possible resource snapshot J' as follows:

- for each time point before s, the entry in J' is the same as in J;
- for each time point x between s and $s + T(b)$ (when the new activity is running), the value of $J[x, y]$ is reduced by $\mathcal{Q}(b)[y]$;
- for each time point x above $s+T(b)$ (after the new activity is completed), the value of $J[x, y]$ is reduced by $\mathcal{Q}(b)[y]$ for non-renewable resources (i.e., when $y > m'$) and remains unchanged for renewable resources (i.e., when $y \leq m'$).

If any entry of the constructed J' is negative, it is not a resource snapshot and hence not added to $\mathcal{J}_{i+1}$; otherwise J' is added to $\mathcal{J}_{i+1}$.

If the algorithm $\mathcal{A}$ results in a set $\mathcal{J}_{n+1}$ that is non-empty, we can reconstruct a solution from the run of the algorithm by standard means—notably, we select an arbitrary resource snapshot in $\mathcal{J}_{n+1}$ and reverse the run of the algorithm to find a corresponding resource snapshot in $\mathcal{J}_n$ along with a mode and starting time for the n-th activity, and proceed until we find modes and starting times for all activities in A. Since there are no precedence constraints and the entries in the resource snapshots are non-negative by construction and represent the amount of available resources, this results in a valid schedule for $\mathcal{I}$. To complete the correctness argument, it suffices to observe that from any solution (ω, α) for $\mathcal{I}$ we can also straightforwardly construct a run of $\mathcal{A}$ which must result in a non-empty set $\mathcal{J}_{n+1}$—indeed, it suffices to follow the branch of the algorithm that results from using mode $\omega(a_i)$ and time step $\alpha(a_i)$ for each activity in A.

The time complexity of the described procedure can be easily seen to lie in $\mathcal{O}(C_{\max} \cdot (c+1)^{C_{\max} \cdot m})$, which is polynomial in $|\mathcal{I}|$. □

Our last algorithm can be viewed as an extension of Theorem 6 to instances of larger makespan, by replacing the bound on the makespan by a weaker restriction, bounding t. This comes at a cost of requiring a bound on c. We note that while we do not explicitly require these instances to be encoded in unary, the combination of flags in fact implies that the instance cannot contain large numbers.

Theorem 7. MRCPSP$\langle m, c, t, \neg P \rangle$ is in P.

Proof. We may assume w.l.o.g. that the image of $\mathcal{Q}$ is a subset of $[c]^m$ (modes mapped by $\mathcal{Q}$ outside of this range are irrelevant because of resource constraints).

Define the *type* of an activity $a_i \in A$, denoted $\tau(a_i)$, as $\{ (\mathcal{Q}(b), T(b)) \mid b \in M_i \}$. Observe that the property of having the same type describes an equivalence relation between activities, which has at most $2^{c^m \cdot t}$ many equivalence classes, each of which we refer to as an *activity type*. Let $\mathcal{T}$ be the set of non-empty activity types.

Moreover if there is a solution, there is a solution (ω, α) such that $\max_{i \in [n]} \alpha(a_i) + T(\omega(a_i)) \leq t \cdot n$. Indeed, if $C_{\max} \leq t \cdot n$ this trivially follows from the makespan constraints, otherwise given a solution (ω, α), $(\omega, a_i \mapsto (i-1) \cdot t)$ is also a solution. Since there are no precedence constraints, any activity with a mode b with $T(b) \leq C_{\max}$ which requires no resources can be trivially scheduled to start at time point 0 using mode b in a solution whenever some solution exists. In such a solution at any time point between 0 and $t \cdot n$ at most $c \cdot m$ of the remaining activities are being

processed concurrently as they have to be assigned to modes using at least some resource.

For the remaining activities we build up partial solutions $((\omega', \alpha')$ where ω' and α' are defined on a subset of A instead of A satisfying all constraints on that subset) along the time steps. We do so by backtracking on the choice of a multiset of at most $c \cdot m$ activity types and modes conforming to these activity types such that activities of these type may be scheduled using these modes in each time step. More formally, we iterate through $i = 0 \dots \min\{t \cdot n, C_{\max}\} - 1$. Within this iteration we iterate through the activity types (with multiplicities) that can be scheduled at time step i. To determine these activity types and their multiplicities we maintain, for each partial solution constructed in each iteration, the resource snapshot $J \in ([c] \cup \{0\})^{C_{\max} \cdot m}$ (defined as in the proof of Theorem 6) induced by this partial solution and a vector $s \in \bigtimes_{\tau \text{ activity type}} ([|\tau|] \cup \{0\})$, describing how many activities of each activity type are not yet in the domain of the partial solution. Initially, i.e., when considering the empty schedule as starting partial solution, the resource snapshot is given by the zero-matrix, and s is given by the vector with an entry $|\tau|$ for each activity type τ. Now a mutiset $\{\tau_1, \dots, \tau_z\}$ of $z \leq c \cdot m$ activity types, can be scheduled at time step i if the multiplicity with which each activity type occurs in the multiset is bounded by the corresponding entry in s and additionally there are $(Q_j, T_j) \in \tau_j$ such that subtracting all Q_j from the $i+1$-th through $i+1+T_j$-th rows of J does not result in negative entries in J. For each such choice of $\{(Q_j, T_j) \in \tau_j \mid j \in [z]\}$, we find an unscheduled activity a with $\tau(a) = \tau_j$ and can set $\omega(a) = b$ such that $(\mathcal{Q}(b), T(b)) = (Q_j, T_j)$ and $\alpha(a) = i$. Appropriate modifications for J and s are straightforward. If we complete iteration $\min\{t \cdot n, C_{\max}\} - 1$ without having scheduled all activities in any encountered solution, we can conclude that no schedule for the instance exists.

The described approach is an iterative branching procedure which is exhaustive modulo activity type equivalence. Hence correctness follows from the fact that activities of the same type can easily be interchanged in a schedule by an easy transformation. The complexity lies in $\mathcal{O}((\min\{t \cdot n, C_{\max}\} - 1) \cdot (2^{c^m \cdot t} \cdot |B|)^{c \cdot m} \cdot |\mathcal{I}|) \subseteq \mathcal{O}(2^{c^{m+1} \cdot m \cdot t} \cdot |\mathcal{I}|^{c^m \cdot t + 2})$. □

Lower Bounds

We now turn towards hardness results for fragments of MRCPSP. First, we state a few previously known lower bounds:

Fact 8 (Uetz (2011), Lemma 5.1.1 via a reduction from 3-COLORING)**.** RCPSP$\langle c, r_{\deg}, t, \neg P, C_{\max}, U\rangle$ is NP-hard.

Fact 9 (Blazewicz, Lenstra and Kan (1983), Theorem 7 via a reduction from 3-PARTITION)**.** RCPSP$\langle m, c, t, S, U\rangle$ is NP-hard.

The third and last known NP-hardness result that we need concerns the fragment RCPSP$\langle m, c, S, \neg P, U\rangle$. Du and Leung (1989, Theorem 2) proved that a scheduling problem equivalent to this fragment is NP-hard (one merely needs to represent the identical machines used in their reduction by capacity units of a single resource; see also (Eyraud-Dubois, Mounie, and Trystram 2007)).

Fact 10. RCPSP$\langle m, c, S, \neg P, U\rangle$ is NP-hard.

Moreover, it is easy to observe that a trivial reduction from BIN PACKING (Garey and Johnson 1979) yields:

Observation 11. RCPSP$\langle m, t, S, \neg P, U\rangle$ is NP-hard.

Proof. We give a polynomial-time reduction from the strongly NP-hard BIN PACKING problem.

Consider an instance of BIN PACKING with bin size w. We construct an instance $\mathcal{I}$ of RCPSP with a single resource of capacity w. Now, for each item of size s we create an activity which that requires s units of this resource and completes in 1 time unit. It is easy to see that the bin packing instance allows a solution with k bins if and only if the RCPSP-instance given by the described A, T, R, Q and $C_{\max} = k$ as a solution for the BIN PACKING instance corresponds to one for $\mathcal{I}$ by scheduling all activities whose respective items are in bin i can be scheduled to start at time point i and vice versa. □

Our following three new reductions show that some severely restricted fragments of (M)RCPSP remain NP-hard andcomplete the complexity map for MRCPSP in terms of the considered explicit restrictions.

Theorem 12. RCPSP$\langle c, r_{\deg}, t, S, C_{\max}, U\rangle$ is NP-hard.

Proof. We give a polynomial-time reduction from 3-SAT by constructing an instance $\mathcal{I}$ from a 3-CNF formula F as follows.

$\mathcal{I}$ has $C_{\max} = 3$ and all processing times of activities 1. For each variable x in F we create a resource r_x with capacity one and two activities x_T, x_F, each requiring one of r_x. Moreover, for each clause C we create a resource r_C with capacity 3, and for each literal ℓ appearing in C we create one activity C_ℓ which requires one r_C. If $\ell = x$ for some variable x (i.e., ℓ is a positive literal), then we create the precedence constraint requiring C_ℓ to start after x_T is complete; otherwise we create the precedence constraint requiring C_ℓ to start after x_F is complete for the respective variable x.

For each clause C, we create three additional auxiliary activities C_0, C_1 and C_2, where $C_0 <_P C_1 <_P C_2$ and which require 0, 2 and 1 resources of type r_C, respectively. This completes our construction.

To complete the proof, we claim that $\mathcal{I}$ is a YES-instance if and only if F is satisfiable. For the backward direction, we can construct a solution from a satisfying assignment for F as follows:

1. If a variable x is set to true, we start x_T at time 0 and x_F and time 1, and otherwise we start x_F at time 0 and x_T at time 1.
2. For each clause C, activities C_0, C_1 and C_2 are scheduled to start at time 0, 1 and 2, respectively (this is necessary for any solution of $\mathcal{I}$).
3. For each clause C, we choose one literal ℓ satisfying C and start C_ℓ at time 1; the activities corresponding to all other literals in C then start at time 2.

It is easy to verify that the above is a solution to $\mathcal{I}$. For the forward direction, consider a solution (ω, α) to $\mathcal{I}$. Observe that due to the makespan restriction, α must always assign C_0, C_1 and C_2 as per Point 2 above, and that w.l.o.g. we

may assume that all activities x_T, x_F are assigned as per Point 1 above (if not, we may adjust α by moving their start times earlier). Consider a variable assignment which sets a variable x to true iff α assigns x_T to start at time 0. Then for each clause C, α must assign one variable of the form C_ℓ to start at time 1 (due to r_C having only 3 available resources and C_2 requiring one); due to the precedence constraints this implies that ℓ satisfies C, and so the result is a satisfying assignment for F. □

Theorem 13. RCPSP$\langle m, t, S, C_{\max}, U\rangle$ is NP-hard.

Proof. We give a polynomial-time reduction from the NP-hard CLIQUE problem, which asks whether a given graph contains a clique of a certain size. Given a graph $G = (V, G)$ and a natural number k (i.e., the desired clique size), we construct an RCPSP-instance $\mathcal{I}$ as follows. We create an activity a_v for every vertex v of G and an activity a_{vw} for every edge vw of G, and we then set $a_v <_P a_{vw}$ and $a_w <_P a_{vw}$. All activities require one time step to process, and we fix $C_{\max} = 3$.

The idea of the set-up we describe in the following is to restrict the activities that can be scheduled to start at the first time step to exactly k activities that correspond to vertices. These k vertices should correspond to the vertices of a clique in G. Then in the next time step, the $\frac{k\cdot(k-1)}{2}$ activities corresponding to the edges of that clique and the remaining vertex activities can be scheduled, allowing for the remaining edge activities to be scheduled in the last time step that $C_{\max} = 3$ allows for. For this to work, all we need to do is to restrict the number of vertex activities that can be scheduled to start in the first time point to be exactly k, the number of vertex activities that can be scheduled to start in the second time point to be exactly $|V| - k$ and the number of edge activities that can be scheduled to start in the third time point to be exactly $|E| - \frac{k\cdot(k-1)}{2}$.

We do this by setting up the resources and resource requirements as follows and introducing 'filler' activities. We consider two resources r_1 and r_2 with capacities $|V|$ and $|E|$, respectively. Each a_v will require one of r_1 and each a_{vw} will require one of r_2. We introduce three further activities a_1, a_2, a_3, each requiring one time step, and set $a_1 <_P a_2 <_P a_3$; a_1 requires $|V| - k$ of r_1, a_2 requires k of r_1, a_3 requires $\frac{k\cdot(k-1)}{2}$ of r_2.

At this point, and given the idea above, it is easy to verify that $\mathcal{I}$ has a solution if and only if G has a clique of size k, as a solution for $\mathcal{I}$ infers a clique of size k by the vertices corresponding to activities scheduled to start in the first time step, and conversely a clique of size k allows for the scheduling of all the activities corresponding to its vertices in the first time step, all the remaining vertices and the clique edges in the second time step, and the remaining edges in the third time step. □

Theorem 14. RCPSP$\langle m, t, S, \neg P, C_{\max}\rangle$ is NP-hard.

Proof. This time, we start from the weakly NP-hard PARTITION problem (Garey and Johnson 1979): decide whether a given multiset $S = \{m_1, m_2, \ldots, m_n\}$ of positive integers such that $\sum_{m_i\in S} = 2b$ can be partitioned into two subsets S_1, S_2 such that $\sum_{m_i\in S_1} = \sum_{m_i\in S_2} = b$.

Given an instance of PARTITION as described above, we create an instance $\mathcal{I}$ of RCPSP with a single resource of capacity b. For each number $m_i \in S$, we now create an activitiy a_i with duration 1 which requires m_i-many units of our resource. It is now easy to see that the PARTITION instance has a solution if and only if $\mathcal{I}$ has a makespan of 2: indeed, there is a one-to-one correspondence between the activities scheduled at time 0 (in a schedule with makespan 2) and the numbers assigned to S_1 (in a solution to PARTITION). □

Summary and Discussion

Altogether, a complete enumeration of all combinations of flags together with two easy inference rules that leave polynomial-time solvability invariant, namely that in simple instances without precedence constraints one can assume m to be bounded (see the argument used for Corollary 3) and in instances with bounded $C_{\max}$ one can assume t to be bounded[3], can be used to ascertain that from the 6 polynomial-time algorithms, we obtain a total of 736 polynomially-tractable fragments of MRCPSP. Similarly, the 7 hardness proofs give rise to a total of 288 NP-hard fragments of MRCPSP. This completely settles the complexity of all fragments of the problem defined in terms of the 10 considered flags.

Solving (M)RCPSP via Structural Restrictions

Here, we use the structure of interactions between activities and resources to push beyond the frontiers of tractability delimited by the complexity map based on explicit restrictions of instance parameters. As mentioned in the introduction, this approach has been very successful for many other prominent problems, and we believe it is highly promising also for (M)RCPSP. However, due to the sheer volume of possible cases and fragments to consider, the two results presented in this section should be viewed primarily as a "proof of concept" and, perhaps, the tip of a (potentially very large) iceberg. Indeed, a thorough investigation of how the structure of activity-resource interactions can be algorithmically exploited is beyond the scope of this work.

For this section it will be useful to recall the definitions of activity- and resource graphs.The first result we present here is a polynomial-time algorithm for all unary MRCPSP instances whose activity graphs have bounded treewidth:

Theorem 15. *There is an algorithm which solves an instance $\mathcal{I}$ of* MRCPSP$\langle U\rangle$ in time at most $\mathcal{O}(|\mathcal{I}|^{5\mathrm{tw}(G_{\mathcal{I}})})$.

We note that Theorem 15 is a generalization of Observation 2 when dealing with unary instances, since the activity graphs of instances in the MRCPSP$\langle n, U\rangle$ fragment have boundedly-many vertices. Similarly, each connected component in the activity graph of an instance in MRCPSP$\langle r_{\deg}, S, \neg P, U\rangle$ has boundedly-many vertices, and so the result also generalizes Corollary 3 in the unary setting.

Proof of Theorem 15. We begin by computing a nice tree-decomposition $\mathcal{T} = (T, \mathcal{X})$ of width $k = \mathrm{tw}(G_{\mathcal{I}})$ (Arnborg,

[3] An automatic checker on this basis is implemented in supplementary file checker.py.

Corneil, and Proskurowski 1987). Let a configuration $\beta(t)$ of a node t in T be a tuple $(\texttt{Mode}, \texttt{Time}, \texttt{Mkspan})$ where

- $\texttt{Mode}$ is a mapping from each activity $a_i \in X_t$ to a mode in M_i,
- $\texttt{Time}$ is a mapping from X_t to $[C_{\max} - 1]$, and
- $\texttt{Mkspan}$ is an integer from $[C_{\max}]$.

Intuitively, we will use configurations to store one possible way of assigning the modes and starting times of activities in X_t that allows us to schedule all activities in $\mathcal{X}^{\downarrow}(t)$ in order to achieve a makespan of $\texttt{Mkspan}$. Let the record $\mathcal{R}(t)$ of a node t in T be the set of all admissible configurations of t, i.e., $(\texttt{Mode}, \texttt{Time}, \texttt{Mkspan}) \in \mathcal{R}(t)$ if and only if there exists an assignment (ω', α') of the activities in $\mathcal{X}^{\downarrow}(t)$ with makespan $\texttt{Mkspan}$ such that $\texttt{Mode}$ is the restriction of ω to X_t and $\texttt{Time}$ is the restriction of α to X_t.

We proceed with some basic observations about our records. Firstly, since the total number of configurations is upper-bounded by ${C_{\max}}^{k+1} \cdot |B|^k$ and the instance is unary, $|\mathcal{R}(t)|$ is polynomial. Secondly, it is easy to compute $\mathcal{R}(t)$ for any leaf t of T by brute-forcing over all assignments and modes of the single activity in that leaf. Thirdly, $\mathcal{I}$ is a YES-instance if and only if the record $\mathcal{R}(r)$ for the root r is non-empty—and moreover, in this case it is easy to reconstruct a solution to $\mathcal{I}$ by backtracking from the root r to determine which entries in the records lead to a non-empty $\mathcal{R}(r)$. Hence, in order to complete the proof it suffices to show how to compute the records for forget, join and introduce nodes.

If t is a forget node with child t' such that $X_{t'} \setminus X_t = \{a_i\}$, then for each configuration $\beta(t') \in \mathcal{R}(t')$ we compute a configuration $\beta(t)$ by removing a_i from the two mappings in $\beta(t')$. We add each such computed configuration to $\mathcal{R}(t)$.

If t is an introduce node with child t' such that $X_t \setminus X_{t'} = \{a_i\}$, then we branch over all mappings $\omega^*(a_i) \in M_i$ and $\alpha^*(a_i) \in [C_{\max} - 1]$. In each branch and for each record $(\texttt{Mode}, \texttt{Time}, \texttt{Mkspan}) \in \mathcal{R}(t')$, we check whether the instance contains sufficient resources for the activities in X_t to be scheduled at times $\alpha^* \cup \texttt{Time}$ and in modes $\omega^* \cup \texttt{Mode}$. (It is useful to note here that since $G_{\mathcal{I}}$ contains no edge between a_i and forgotten vertices (i.e., vertices in $\mathcal{X}^{\downarrow}(t) \setminus X_t$), a_i can never be "in conflict" with any such forgotten activity.) If this is the case, then we check when a_i ends based on the current choice of α^* and ω^*, and we update $\texttt{Mkspan}$ accordingly (naturally, we discard records where a_i ends after $C_{\max}$). We then add the configuration with the new $\texttt{Mkspan}$ and with the mappings $\omega^* \cup \texttt{Mode}$, $\alpha^* \cup \texttt{Time}$ to $\mathcal{R}(t)$.

If t is a join node with children t', t'', then we loop over each configuration $(\texttt{Mode}', \texttt{Time}', \texttt{Mkspan}') \in \mathcal{R}(t')$ and compare it to each configuration $(\texttt{Mode}'', \texttt{Time}'', \texttt{Mkspan}'') \in \mathcal{R}(t'')$. Whenever $\texttt{Mode}' = \texttt{Mode}''$ and $\texttt{Time}' = \texttt{Time}''$, we add the configuration $(\texttt{Mode}', \texttt{Time}', \max(\texttt{Mkspan}', \texttt{Mkspan}''))$ to $\mathcal{R}(t)$.

The runtime of these steps is dominated by the runtime of the join node, which requires time at most ${C_{\max}}^{k+1} \cdot |B|^k)^2 \cdot |\mathcal{I}|$. Hence, the total runtime of the algorithm is upper-bounded by $|\mathcal{I}|^{5k}$. □

Our final, perhaps most technically challenging, result is an algorithm to solve MRCPSP$\langle \neg P, C_{\max}, U \rangle$ in polynomial time when the treewidth of the resource graph $H_{\mathcal{I}}$ is bounded. This result is a strict generalization of Theorem 6.

Theorem 16. *There is an algorithm which solves an instance $\mathcal{I}$ of* MRCPSP$\langle U \rangle$ in time at most $|\mathcal{I}|^{C_{\max} \cdot 2^{\mathcal{O}(\mathrm{tw}(G_{\mathcal{I}}))}}$.

Proof. On a high level, the algorithm is based on a combination of the technique used in the proof of Theorem 6 and leaves-to-root dynamic programming as in the proof of Theorem 15.

We begin by once again computing a nice tree-decomposition $\mathcal{T} = (T, \mathcal{X})$ of width $k = \mathrm{tw}(G_{\mathcal{I}})$. For a node t of T, we let a *t-resource snapshot* be a $C_{\max} \times |X_t|$ matrix over $[c]$ with columns indexed by the elements of X_t; observe that the number of t-resource snapshots is upper-bounded by $(c+1)^{C_{\max} \cdot (k+1)}$. The t-resource snapshot of a schedule (ω, α) for a subset A' of activities in A is the restriction of the resource snapshot of (ω, α) for A' to the resources (i.e., columns) in X_t.

These t-resource snapshots serve the same purpose as resource snapshots in the proof of Theorem 6. Because of the properties of a tree decomposition it will be sufficient to consider these restricted snapshots. We consider t-resource snapshots because, in contrast to the situation of Theorem 6, we cannot bound the number of resource snapshots but the bounded width of the tree decomposition immediately infers a bound on the number of t-resource snapshots.

An further complication that arises here and was not present in the setting of Theorem 6 is that we are given precedence constraints.

For a set $L \subseteq R$ of resources, let A_L be the restriction of A to those activities whose modes only use resources from L; formally, $A_L = \{\, a_i \mid \forall b \in M_i, r_j \notin L \quad \mathcal{Q}(b)[j] = 0 \,\}$. Define A_t and $A_t^{\downarrow}$ as shorthand for A_{X_t} and $A_{X_t^{\downarrow}}$. Note that, by definition of $H_{\mathcal{I}}$ and tree decompositions, for each activity $a \in A$ there is at least one node t such that $a \in A_t$.

We can now give a high-level overview of the algorithm. At its core, the algorithm still aims at computing a resource snapshot for all activities and resources (corresponding to the set $\mathcal{J}_{n+1}$ in Theorem 6). However, now it never considers the "full" resource snapshots, but instead only the snapshots for resources that appear in the current bag t (i.e., the t-resource snapshots), and these snapshots are computed only for activities which use the resources that have appeared so far (i.e., activities in $A_t^{\downarrow}$). On its own, defining records in this way would be sufficient for a dynamic programming algorithm as long as there were no *join* nodes (i.e., for a path-decomposition); unfortunately, to correctly handle *join* nodes (and in particular, to prevent double counting) one needs to store a second kind of record which details the t-resource snapshots for A_t only. These secondary components of our records then present an additional challenge for handling *forget* nodes—to update them, one needs to store not only a t-resource snapshot for A_t, but a t-resource snapshot for $A_{X'}$ for every $X' \subseteq X_t$.

With these challenges in mind, we can proceed to formalizing the algorithm. Let the record $\mathcal{J}_t$ of a node t be the set of all tuples (J, γ) such that there is a schedule *sched* $= (\omega, \alpha)$

for the activities in $A_t^{\downarrow}$ such that J is the t-resource snapshot of *sched*, and γ is a mapping from each $X' \subseteq X_t$ to the t-resource snapshot of the restriction of *sched* to the activities in X'.

For convenience, we enhance our records by a mapping `wit` from each tuple in $\mathcal{J}_t$ to an (arbitrary) witness schedule for $A_t^{\downarrow}$ that satisfies the properties of a corresponding schedule *sched*. For each leaf t of T, $\mathcal{J}_t$ can be computed by simply invoking (the proof of) Theorem 6 on the activities in A_t. Moreover, $\mathcal{I}$ is a YES-instance if and only if the root r satisfies $\mathcal{J}_r \neq \emptyset$, as witnessed by the witness schedule for an arbitrary entry in $\mathcal{J}_r$. Hence, to complete the proof it suffices to show how to compute $\mathcal{J}_t$ in a leaves-to-root manner as in Theorem 15.

If t is a forget node with child t' such that $X_{t'} \setminus X_t = \{r_i\}$, then from each tuple $(J, \gamma) \in \mathcal{J}_{t'}$, we prune the mapping γ by removing all subsets of $X_{t'}$ containing r_i, and then for J and all remaining images of γ we remove the column corresponding to r_i. We add each tuple resulting from these operations into $\mathcal{J}_t$. The only change to `wit` is that if several tuples in $\mathcal{J}_{t'}$ merge into a single tuple in $\mathcal{J}_t$, we select the reference solution for the new tuple arbitrarily among the reference solutions for the corresponding tuples in $\mathcal{J}_{t'}$.

If t is a join node with children t', t'', we proceed as follows. First, we exhaustively loop through every pair of entries $(J', \gamma') \in \mathcal{J}_{t'}$ and $(J'', \gamma'') \in \mathcal{J}_{t''}$, and check if $\gamma' = \gamma''$ (if not, we discard and proceed to the next pair). Next, we compute an entry (J, γ) by setting $\gamma = \gamma'$ and $J = J' + J'' - \gamma(X_t)$ (the subtraction deals with double-counting of activities in the intersection). We compute $\texttt{wit}(J, \gamma)$ by assigning all activities in $A_{t'}^{\downarrow} \setminus A_{t'}$ in the same way as $\texttt{wit}'(J', \gamma')$, all activities in $A_{t''}^{\downarrow} \setminus A_{t''}$ in the same way as $\texttt{wit}''(J'', \gamma'')$, and all activities in $A_{t'} = A_{t''}$ in the same way as either $\texttt{wit}'(J', \gamma')$ or $\texttt{wit}''(J'', \gamma'')$ (the choice here does not matter, since both options result in the same t-resource snapshots in γ).

If t is an introduce node with child t' such that $X_t \setminus X_{t'} = \{r_i\}$, we proceed as follows. Let $A' = A_t \setminus A_{t'}^{\downarrow}$ be the set of activities that need to be processed in order to compute the t-resource snapshots. We compute these snapshots by expanding on the dynamic programming algorithm of Theorem 6; instead of reproving the whole (expanded) statement, we build on that algorithm and provide an overview of the required modifications.

For each entry $(J', \gamma') \in \mathcal{J}_{t'}$, set $J_0 = J'$ and set $\gamma_0 = \gamma'$. Now, for an activity a in A', we branch on its starting time and mode (in the same way as in Theorem 6). We then check if a can be scheduled into J_0 without violating the resource and makespan constraints, and if so, we expand $\texttt{wit}'(J', \gamma')$ by scheduling a in this way and alter J_0 by subtracting the appropriate amount of resources. The same changes as in J_0 are then performed on all t-resource snapshots $\gamma_0(X')$ such that $X' \subseteq X_t$ and (all modes of) a only uses resources from X'. This results in a tuple (J_1, γ_1), and we iterate this process (by selecting the next activity in A') until we obtain the final tuple $(J_{|A'|}, \gamma_{|A'|})$, which we then add into $\mathcal{J}_t$. Correctness follows by repeating the same arguments as in Theorem 6.

The size of our records is upper-bounded by $(c+1)^{C_{\max}\cdot(k+1)} \cdot ((c+1)^{C_{\max}\cdot(k+1)})^{2^k} \cdot |\mathcal{I}| \in \mathcal{O}(|\mathcal{I}|^{C_{\max}\cdot 2^{2k}})$. The runtime of our dynamic programming steps is dominated by the runtime of the join node, which needs to loop over all pairs of records in the two children. Hence the total runtime of the algorithm is in $|\mathcal{I}|^{C_{\max}\cdot 2^{\mathcal{O}(k)}}$. □

Concluding Remarks

We introduced a series of new algorithmic upper and lower bounds that together paint a complete picture of the classical complexity of (M)RCPSP in terms of explicit restrictions on its instances. An extension of RCPSP which we did not directly address in this work is RCPSP/max, where instead of simple precedence constraints one can specify a desired maximum and minimum time gap between finishing one and starting another activity. Naturally, all our lower bounds also carry over to this more general problem. Moreover, the three positive results for fragments with precedence constraints (Observation 2, Theorem 4 and Corollary 5) extend to the RCPSP/max setting with almost no changes to their proofs.

It would be interesting to refine the obtained complexity map from the *parameterized complexity* viewpoint (Downey and Fellows 2013; Cygan et al. 2015). In particular, most of the tractability results presented in this paper do not readily translate to *fixed-parameter tractability*, and it would certainly be worthwhile to determine which parameterizations of (M)RCPSP give rise to fixed-parameter algorithms.

Finally, we also indicated how one can algorithmically exploit the structural properties of activity and resource interactions through the use of graph representations and structural parameters. We believe this is a promising direction for future research.

References

[Arnborg, Corneil, and Proskurowski 1987] Arnborg, S.; Corneil, D. G.; and Proskurowski, A. 1987. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods* 8(2):277–284.

[Artigues, Demassey, and Neron 2008] Artigues, C.; Demassey, S.; and Neron, E. 2008. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. ISTE/Wiley.

[Barrios, Ballestín, and Valls 2011] Barrios, A.; Ballestín, F.; and Valls, V. 2011. A double genetic algorithm for the mrcpsp/max. *Computers & OR* 38(1):33–43.

[Blazewicz, Lenstra, and Kan 1983] Blazewicz, J.; Lenstra, J. K.; and Kan, A. H. G. R. 1983. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics* 5(1):11–24.

[Bodlaender et al. 2016] Bodlaender, H. L.; Drange, P. G.; Dregi, M. S.; Fomin, F. V.; Lokshtanov, D.; and Pilipczuk, M. 2016. A c$^{\text{k}}$ n 5-approximation algorithm for treewidth. *SIAM J. Comput.* 45(2):317–378.

[Bofill et al. 2017] Bofill, M.; Coll, J.; Suy, J.; and Villaret, M. 2017. An efficient SMT approach to solve mrcpsp/max

instances with tight constraints on resources. In *CP 2017*, 71–79.

[Brucker and Knust 2011] Brucker, P., and Knust, S. 2011. *Complex Scheduling*. Springer, 2nd edition.

[Cygan et al. 2015] Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer.

[Damay, Quilliot, and Sanlaville 2007] Damay, J.; Quilliot, A.; and Sanlaville, E. 2007. Linear programming based algorithms for preemptive and non-preemptive RCPSP. *European Journal of Operational Research* 182(3):1012–1022.

[Downey and Fellows 2013] Downey, R. G., and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

[Du and Leung 1989] Du, J., and Leung, J. Y. 1989. Complexity of scheduling parallel task systems. *SIAM J. Discrete Math.* 2(4):473–487.

[Eyraud-Dubois, Mounie, and Trystram 2007] Eyraud-Dubois, L.; Mounie, G.; and Trystram, D. 2007. Analysis of scheduling algorithms with reservations. In *21th International Parallel and Distributed Processing Symposium (IPDPS 2007), Proceedings, 26-30 March 2007, Long Beach, California, USA*, 1–8.

[Freuder 1982] Freuder, E. C. 1982. A sufficient condition for backtrack-free search. *J. ACM* 29(1):24–32.

[Fu, Varakantham, and Lau 2010] Fu, N.; Varakantham, P.; and Lau, H. C. 2010. Towards finding robust execution strategies for rcpsp/max with durational uncertainty. In *ICAPS 2010*, 73–80.

[Fu, Varakantham, and Lau 2016] Fu, N.; Varakantham, P.; and Lau, H. C. 2016. Robust partial order schedules for rcpsp/max with durational uncertainty. In *ICAPS 2016*, 124–130.

[Ganian and Ordyniak 2018] Ganian, R., and Ordyniak, S. 2018. The complexity landscape of decompositional parameters for ILP. *Artif. Intell.* 257:61–71.

[Ganian, Ordyniak, and Ramanujan 2017] Ganian, R.; Ordyniak, S.; and Ramanujan, M. S. 2017. Going beyond primal treewidth for (M)ILP. In *AAAI 2017*, 815–821.

[Garey and Johnson 1975] Garey, M. R., and Johnson, D. S. 1975. Complexity results for multiprocessor scheduling under resource constraints. *SIAM J. Comput.* 4(4):397–411.

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

[Gehrke et al. 2018] Gehrke, J. C.; Jansen, K.; Kraft, S. E. J.; and Schikowski, J. 2018. A PTAS for scheduling unrelated machines of few different types. *Int. J. Found. Comput. Sci.* 29(4):591–621.

[Gottlob, Scarcello, and Sideri 2002] Gottlob, G.; Scarcello, F.; and Sideri, M. 2002. Fixed-parameter complexity in AI and nonmonotonic reasoning. *Artif. Intell.* 138(1-2):55–86.

[Kloks 1994] Kloks, T. 1994. *Treewidth, Computations and Approximations*, volume 842 of *LNCS*. Springer.

[Kolisch and Padman 2001] Kolisch, R., and Padman, R. 2001. An integrated survey of deterministic project scheduling. *Omega* 29(3):249–272.

[Kuster, Jannach, and Friedrich 2007] Kuster, J.; Jannach, D.; and Friedrich, G. 2007. Handling alternative activities in resource-constrained project scheduling problems. In *IJCAI 2007*, 1960–1965.

[Marx 2010] Marx, D. 2010. Can you beat treewidth? *Theory of Computing* 6(1):85–112.

[Poppenborg and Knust 2016] Poppenborg, J., and Knust, S. 2016. Modeling and optimizing the evacuation of hospitals based on the MRCPSP with resource transfers. *EURO J. Computational Optimization* 4(3-4):349–380.

[Robertson and Seymour 1983] Robertson, N., and Seymour, P. D. 1983. Graph minors. i. excluding a forest. *J. Comb. Theory, Ser. B* 35(1):39–61.

[Samer and Szeider 2009] Samer, M., and Szeider, S. 2009. Fixed-parameter tractability. In Biere, A.; Heule, M.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*. IOS Press. 425–454.

[Samer and Szeider 2010] Samer, M., and Szeider, S. 2010. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.* 76(2):103–114.

[Schwindt and Zimmermann 2015] Schwindt, C., and Zimmermann, J. 2015. *Handbook on Project Management and Scheduling Vol.1*. Springer.

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

[Smith and Pyle 2004] Smith, T. B., and Pyle, J. M. 2004. An effective algorithm for project scheduling with arbitrary temporal constraints. In *AAAI 2004*, 544–549.

[Song 2017] Song, W. 2017. Project scheduling in complex business environments. In *AAAI 2017*, 5052–5053.

[Thorup 1998] Thorup, M. 1998. All structured programs have small tree-width and good register allocation. *Inf. Comput.* 142(2):159–181.

[Uetz 2011] Uetz, M. 2011. *Algorithms for Deterministic and Stochastic Scheduling*. Ph.D. Dissertation.

[van Bevern et al. 2016] van Bevern, R.; Bredereck, R.; Bulteau, L.; Komusiewicz, C.; Talmon, N.; and Woeginger, G. J. 2016. Precedence-constrained scheduling problems parameterized by partial order width. In *DOOR 2016*, 105–120.

[Varakantham, Fu, and Lau 2016] Varakantham, P.; Fu, N.; and Lau, H. C. 2016. A proactive sampling approach to project scheduling under uncertainty. In *AAAI 2016*, 3195–3201.