The Complexity Landscape of Resource-Constrained Scheduling

Robert Ganian\textsuperscript{1}, Thekla Hamm\textsuperscript{1} and Guillaume Mescoff\textsuperscript{2}
\textsuperscript{1}Vienna University of Technology
\textsuperscript{2}Rennes University
rganian@gmail.com, thamm@ac.tuwien.ac.at, guillaume.mescoff@ens-rennes.fr

Abstract

The Resource-Constrained Project Scheduling Problem (RCPSP) and its extension via activity modes (MRCPS) 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 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 the complexity of interactions between individual activities. In particular, we show that if the treewidth of a graph which captures such interactions is bounded by a constant, then we can solve MRCPS in polynomial time.

1 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 et al., 2007; Varakantham et al., 2016; Song, 2017], Operations Research and Scheduling [van Bevern et al., 2016; Fu et al., 2010; Fu et al., 2016]; see also the survey [Kolisch and Padman, 2001] and book [Artigues et al., 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 [Schwindt and Zimmermann, 2015].

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 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 that activities also adhere to a precedence order.

A prominent generalisation of RCPSP that has received considerable attention [Bofll et al., 2017; Barrios et al., 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 (MRCPS).

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 combinations of variants arising from the following explicit restrictions/attributes which are immediately tied to numerical properties of the input or have been established in previous literature:

- 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 \((C_{\text{max}})\), and/or on the number of activities that can use each resource \((r_{\text{deg}})\):
- No precedence constraints \((-P)\);
- "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 \((\text{MRCSP})\) or not \((\text{RCPSP})\);
- 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, \(\text{MRCPSP}(c, r_{\text{deg}})\) refers to instances of \(\text{MRCSP}\) where each resource has capacity bounded (by a constant), and each resource is only used by a \((\text{constant})\)-bounded number of activities.

Each of the above attributes can be viewed as an independent binary “switch”; altogether this amounts to \(2^{10}\) considered fragments of \((\text{M})\text{RCSP}\). 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 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. 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 concept of treewidth. As our second contribution, we show that treewidth allows us to push the frontiers of tractability for \(\text{MRCPSP}\) when applied to the activity graph—a graph which represents activities as vertices and adds edges between activities which interact either by sharing a resource or a precedence constraint.

**Related work.** While the treewidth of instances has not been considered for \(\text{RCPSP}\) yet, the parameter has found numerous applications in prominent subfields and problems that are relevant for AI research, such as SAT [Gottlob et al., 2002], ILP [Ganian and Ordyniak, 2018] and CSP [Cohen et al., 2015]. 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].

RCSP is known to be polynomial-time solvable when the poset width of the precedence constraints is bounded [van Bevern et al., 2016].

## 2 Preliminaries

For an integer \(i\), we use \([i]\) as shorthand for \(\{1, \ldots, i\}\). The function \(\text{argmin}\) refers to an (arbitrary) argument of the minimum. We assume that \(\mathbb{N}\) is the set of non-negative integers. For a vector \(R\), we use \(R[\ell]\) to denote its \(\ell\)-th coordinate.

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

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

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

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

**Precedence constraints:** For each activity \(a_i, a_i' \in A\) such that \(a_i <_P a_i'\): \(\alpha(a_i) + T(w_i) \leq \alpha(a_i')\).

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

\(\text{RCPSP}\) is the restriction of \(\text{MRCPSP}\) 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 \(M\) and having \(T\) and \(Q\) directly refer to activities in \(A\).

The problem definition suggests a number of interesting and natural parameters which we wish to consider as flags used to define the basic fragments of \((\text{M})\text{RCSP}\) considered in this paper. A class \(D\) of \(\text{MRCPSP}\) instances has the flag \((n)\) if there exists some integer \(z\) such that each instance in \(D\) has at most \(z\) activities. The flags \((m)\) (total number of resources—renewable as well as non-renewable), \((t)\) (maximum value of \(T\)), \((c)\) (maximum value of \(C\)), \((C_{\text{max}})\) are defined analogously. \(D\) has the flag \((r_{\text{deg}})\) if there exists some integer \(z\) such that, for each instance \(I \in D\) and for each resource \(r_i\) in that instance, there are at most \(z\) activities which can use \(r_i\)—formally, \(|\{a_i : \forall \beta \in M_i, C(b)[\ell] = 0\}| \geq n - z\). Intuitively, \((r_{\text{deg}})\) 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—namely the ones signifying the lack of precedence constraints ($\langle \neg P \rangle$), an unary encoding of the numbers ($\langle U \rangle$), and whether we have modes or not—are self-explanatory. The last remaining flag is $(S)$ (short for “simple”), which signifies that for every activity $a_i$ in an instance in the class $D$, there is at most one resource used by $a_i$ in any mode—formally, $\forall i \in [n] \ | \{ j \in [m] \ | \exists b \in M_t (Q(b)[\ell] > 0) \} \leq 1$. Simple instances represent a middle ground between instances with a single resource and general instances [Damay et al., 2007] and have a natural correspondence to classical scheduling over $m$ types of machines [Gehrke et al., 2018].

We will use $|I|$ to denote the size of a (unary or binary, depending on the flag “$U$”) encoding of the instance $I$.

**Treewidth and Graph Representations.** A nice tree-decomposition $T$ of a graph $G = (V,E)$ is a pair $(T, \mathcal{X})$, where $T$ is a tree rooted at a node $r$ 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 $T$.
- $|X_t| = |X_{t'}| = 1$ for every leaf $t'$ of $T$.
- There are only three kinds of non-leaf nodes in $T$:
  - **Introduce node:** a node $t$ with exactly one child $t'$ such that $X_t = X_{t'} \cup \{ v \}$ for some vertex $v \not\in X_{t'}$.
  - **Forget node:** a node $t$ with exactly one child $t'$ such that $X_t = X_{t'} \setminus \{ w \}$ for some vertex $w \in X_{t'}$.
  - **Join node:** a node $t$ with two children $t_1, t_2$ such that $X_t = X_{t_1} \cup X_{t_2}$.

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

For every fixed $k$, a nice tree-decomposition of a graph $G$ of treewidth $k$ can be computed efficiently if one exists [Bodlaender et al., 2016; Kloks, 1994; Arnborg et al., 1987]. We use $\mathcal{X}^+(t)$ to denote the set of all vertices in bags of the sub-tree of $T$ rooted at $t$.

In our initial analysis of the potential applications of treewidth, we will restrict our attention to a natural graph representation of a MRCPSP instance $I$ which captures how activities may interact with each other. Notably, the activity graph $\mathcal{G}_I$ has vertex set $A$ and edges represent precedences as well as the possibility of using the same resource—in particular, its edge set is $\{ a_i a_j \mid (a_i < P a_j) \lor (a_i < P a_j) \lor (\exists b \in M_i, b' \in M_j, \ell \in [m] Q(b)[\ell] \neq 0 \land Q(b')[\ell] \neq 0) \}$. An illustration is provided in Figure 2.

![Figure 1](image1.png)

![Figure 2](image2.png)

### 3 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).

#### 3.1 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 1.** MRCPSP$(\langle n \rangle)$ is in P.

**Proof Sketch.** Branch over all permutations of the activities and all assignments of activities to their modes. In each branch, greedily build a solution which assigns activities to the selected modes and starts them in an order which does not violate the permutation, or decide that this is not possible, in quadratic time. The time complexity of this procedure lies in $O(n!) \prod_{i=1}^m |M_i| \cdot |I|^2 \subseteq O(|I|^{n+2})$.

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 2. MRCPSP\((r_{\text{deg}}, S, \neg P)\) is in \(P\).

Proof Sketch. 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 \(I\) of MRCPSP\((r_{\text{deg}}, S, \neg P)\) can be split into a set of independent instances, each containing a single resource and at most \(r_{\text{deg}}\) activities. Using the previous observation this yields an algorithm in \(O(m \cdot r_{\text{deg}} \cdot |I|^{r_{\text{deg}}+2}) \subseteq O(|I|^{r_{\text{deg}}+2})\). \(\square\)

We now proceed to fragments with non-trivial algorithms.

Theorem 3. MRCPSP\((m, c, C_{\text{max}})\) is in \(P\).

Proof. Any solution \((\omega, \alpha)\) to an instance \(I\) contains at most \(q = C_{\text{max}} \cdot c \cdot m\) modes which use at least one resource—i.e., the set \(B_{>0} = \{ b \in \omega(A) \mid \exists j \in \mathbb{N} : Q(b)[j] > 0 \}\) 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 \(I\) using the algorithm \(A\) 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 \(\omega\) 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 \(Q(b) = 0^m\) (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 \(\omega\) 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 \(\alpha\) which conforms to this ordering whenever such an \(\alpha\) exists. The caveat here is that this exact approach would introduce a linear dependency on \(n\) which is in general not in \(\text{poly}(|I|)\).

Instead, \(A\) 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 running time of \(A\) can be shown to lie in \(O(|B|^{C_{\text{max}} \cdot c \cdot m} \cdot (C_{\text{max}} \cdot c \cdot m)^2).\) \(\square\)

The proof strategy for Theorem 3 can be combined with that of Observation 1 to obtain a polynomial-time algorithm when \(r_{\text{deg}}\) and \(m\) are bounded. The resulting algorithm runs in time \(O((r_{\text{deg}} \cdot m)! \cdot |I|^{r_{\text{deg}}+2})\).

Corollary 4. MRCPSP\((m, r_{\text{deg}})\) is in \(P\).

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 5. MRCPSP\((m, \neg P, C_{\text{max}}, U)\) is in \(P\).

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

Given an instance \(I\), let \(J_0\) be the set of resource snapshots of all partial schedules for the activities \(\{a_j \mid j \leq i\}\). Clearly, \(J_0\) contains a single resource snapshot, namely the one where \(J[x, y] = C(r_y)\) for all \(x, y\). On the other hand, if \(J_0 \neq \emptyset\) then \(I\) is clearly a YES-instance.

To prove the theorem, we describe a dynamic programming algorithm \(A\) which computes \(J_{i+1}\) from \(J_i\). \(A\) begins by looping over all resource snapshots in \(J_i\), branching over each mode \(b \in M_{i+1}\) of activity \(a_{i+1}\) and branching over each starting time \(s \in [C_{\text{max}} - T(b)]\). For each such choice of resource snapshot \(J, b\) and \(s\), it creates a new possible resource snapshot \(J'\). If any entry of the constructed \(J'\) is negative, it is not a resource snapshot and hence not added to \(J_{i+1}\); otherwise \(J'\) is added to \(J_{i+1}\).

If the algorithm \(A\) results in a set \(J_{n+1}\) that is non-empty, we can reconstruct a solution from the run of the algorithm by standard means; otherwise we conclude that \(I\) is a NO-instance. Note that each combination of mode, starting time and resource snapshot is considered at most once when updating the resource snapshots. Hence time complexity lies in \(O(|B| \cdot C_{\text{max}} \cdot (c + 1)^{C_{\text{max}} \cdot m}) \subseteq O(|I|^{C_{\text{max}} \cdot m+2}).\) \(\square\)

Our last algorithm can be viewed as an extension of Theorem 5 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 \(c\).

Theorem 6. MRCPSP\((m, c, t, \neg P)\) is in \(P\).

Proof Sketch. We may assume w.l.o.g. that the image of \(Q\) is a subset of \([c]^m\) (modes mapped by \(Q\) 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 \(\{ (Q(b), T(b)) \mid b \in M_i \}\). Observe that the property of having the same type describes an equivalence relation between activities, which has at most \(2^m\) many equivalence classes, each of which we refer to as an activity type. Let \(T\) be the set of non-empty activity types.

If there is a solution, there is a solution \((\omega', \alpha')\) such that \(\max_{x \in [n]} \alpha(a_i) + T(\omega(a_i)) \leq t \cdot n\) and any activity with a mode \(b\) with \(T(b) \leq C_{\text{max}}\) which requires no resources is scheduled to start at time 0 using mode \(b\). 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 \(\omega'\) and \(\alpha'\) 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 \(i = 0 \ldots \min\{t \cdot n, C_{\text{max}}\} - 1\). 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 \(J \in ([c] \cup \{0\})^{C_{\text{max}} \cdot m}\) (defined as in the proof of Theorem 5) induced by this partial solution and a vector \(s \in X_t^{\text{activity type}}(\{\tau\} \cup \{0\})\), describing how many activities of each activity type are not yet in the domain of the
partial solution. A multiset \( \{ \tau_1, \ldots, \tau_z \} \) 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 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 \( \{(Q_j, T_j) \in \tau_j | j \in [z]\} \), we find an unsched-
uled activity \( \alpha \) with \( \tau(\alpha) = \tau_j \) and can set \( \omega(\alpha) = b \) such that \( (Q(b), T(b)) = (Q_j, T_j) \) and \( \alpha(a) = i \). Appropriate mod-
ifications 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 exhaustion modulo activity type equivalence. Hence correctness follows from the fact that activities of the same type can be scheduled at time step \( i \) by a systematic transformation. The complexity lies in \( \mathcal{O}(\min(t \cdot n, C_{\max}) - 1) \cdot (2^m \cdot |\mathcal{B}|)^{c \cdot m - 2} \).

### 3.2 Lower Bounds

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

**Fact 7** (Uetz [2011], Lemma 5.1.1). RCPSP\((c, r_{\text{deg}}, t, \neg P, C_{\max}, U)\) is NP-hard.

**Fact 8** (Blazewicz, Lenstra and Kan [1983], Theorem 7). RCPSP\((m, c, t, S, U)\) is NP-hard.

The third and last known NP-hardness result that we need concerns the fragment RCPSP\((m, c, S, \neg P, U)\). Du and Le-
ung (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).

**Fact 9.** RCPSP\((m, c, S, \neg P, U)\) is NP-hard.

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

**Observation 10.** RCPSP\((m, t, S, \neg P, U)\) is NP-hard.

Our following three new reductions complete the complex-
ity map for MRCPS in terms of explicit restrictions.

**Theorem 12.** RCPSP\((m, t, S, C_{\max}, U)\) is NP-hard.

**Proof Sketch.** We give a reduction from 3-SAT by construct-
ing a solution \( \mathcal{I} \) 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 \( \ell \in C \) we create one activity \( C_\ell \) which requires one \( r_C \). If \( \ell = x \) for some variable \( x \) (i.e., \( \ell \) is a posi-
tive literal), we create the precedence constraint requiring \( C_\ell \) to start after \( x_T \) is completed; otherwise we create the prece-
dence constraint requiring \( C_\ell \) to start after \( x_F \) is completed.

For each clause \( C \), we now create three new activities \( c_0, C_1 \) and \( C_2 \), where \( c_0 < r C_1 < r C_2 \) and which require 0, 2 and 1 resources of type \( r_C \), respectively. This completes our construction. To complete the proof, it suffices to verify that \( \mathcal{I} \) is a YES-instance iff \( F \) is satisfiable.

**Theorem 13.** RCPSP\((m, t, S, \neg P, C_{\max})\) is NP-hard.

**Proof.** This time, we start from the weakly NP-hard PAR-
TITION 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 ca-
pacity \( b \). For each number \( m_i \in S \), we now create an activity \( a_i \) 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 iff \( \mathcal{I} \) 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).

### 3.3 Summary and Discussion

Note that the following modifications leave instances invariant with respect to polynomial-time solvability: (1) in simple instances without precedence constraints one can assume \( m \) to be bounded (see the argument used for Corollary 2), and (2) in instances with bounded \( C_{\max} \) one can assume \( t \) to be bounded. These two simple observations together with a complete enumeration of all combinations of flags yield that the 6 polynomial-time algorithms, give rise to a total of 736
fragments of MRCPPSP which are in P. Similarly, the 7 hardness results imply a total of 288 NP-hard fragments of MRCPPSP. This completely settles the complexity of all fragments of the problem defined in terms of the 10 considered flags.

4 Solving (M)RCPSP via Structural Restriction

Here, we use the structure of interactions between activities 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 result 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 instances can be algorithmically exploited is beyond the scope of this work.

Theorem 14. MRCPPSP(U) can be solved in time \( O(|I|^{k(w|G|)}) \), where \( I \) is the input instance.

We note that Theorem 14 is a generalization of Observation 1 when dealing with unary instances, since the activity graphs in the MRCPPSP(n, U) fragment have boundedly-many vertices. Similarly, each connected component in the activity graph in an instance in MRCPPSP(P, \( \exists P, U \)) has boundedly-many vertices, and so the result also generalizes Corollary 2 in the unary setting.

Proof Sketch of Theorem 14. We begin by computing a nice tree-decomposition \( T = (T, X) \) of width \( k = tw(G) \) [Arnborg et al., 1987]. Let a configuration \( \beta(t) \) of a node \( t \) in \( T \) be a tuple \( (Mode, Time, Mkspan) \) where

- \( Mode \) is a mapping from each activity \( a_i \in X_t \) to a mode in \( M_t \);
- \( Time \) is a mapping from \( X_t \) to \( \{C_{max} - 1\}, \) and
- \( Mkspan \) is an integer from \( \{C_{max}\} \).

Intuitively, we use configurations to store possible ways of assigning modes and starting times of activities in \( X_t \) that allow to schedule activities in \( X^t(t) \) with makespan \( Mkspan \). For \( t \in V(T) \) we let the record \( R(t) \) consist of all admissible configurations of \( X^t(t) \), i.e., \( (Mode, Time, Mkspan) \in R(t) \) iff there is an assignment satisfying all precedence and other constraints \( (\omega, \alpha) \) of the activities in \( X^t(t) \) with makespan \( Mkspan \) such that \( Mode \) and \( Time \) are the restrictions of \( \omega \) to \( X_t \) and \( \alpha \) to \( X_t \), respectively.

As the total number of configurations is upper-bounded by \( C_{max}^{k+1} \cdot |B|^k \) and the instance is unary, \( |R(t)| \) is bounded by a polynomial in \( |I| \). Moreover, it is easy to compute \( R(t) \) for any leaf \( t \) of \( T \) by brute-forcing over all assignments and modes of the single activity in that leaf. \( I \) is a YES-instance iff the record \( R(r) \) for the root \( r \) is non-empty—moreover, in this case it is easy to reconstruct a solution to \( I \) by backtracking from the root \( r \) to determine which entries in the records lead to a non-empty \( R(r) \). 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 \( \beta(t') \in R(t') \) we compute a configuration \( \beta(t) \) by removing \( a_i \) from the two mappings in \( \beta(t') \). We add each such computed configuration to \( R(t) \).

If \( t \) is an introduce node with child \( t' \) such that \( X_{t} \setminus X_{t'} = \{a_i\} \), then we branch over all mappings \( \omega^*(a_i) \in M_t \) and \( \alpha^*(a_i) \in [C_{max} - 1] \). In each branch and for each record \( (Mode, Time, Mkspan) \in R(t') \), we check whether the instance contains sufficient resources for the activities in \( X_t \) to be scheduled at times \( \omega^* \cup Time \) and in modes \( \omega^* \cup Mode \). Moreover, we check that \( a_i \) satisfies the precedence constraints w.r.t. the other activities in \( X_t \). If both checks are successful, we add the new \( Mkspan \) and with the mappings \( \omega^* \cup Mode, \alpha^* \cup Time \) to \( R(t) \).

If \( t \) is a join node with children \( t, t' \), then we check compatibility (i.e., equality) of every pair of elements of \( R(t') \) and \( R(t'') \) and add compatible elements to \( R(t) \).

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

5 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 1, Theorem 3 and Corollary 4) extend to the RCPSP/max setting with minimal changes.

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 interactions through the use of graph representations and structural parameters. We believe this is a promising direction for future research; for instance, a similar approach could also be used to consider graph representations of interactions between resources.

Acknowledgments

R. Ganian and T. Hamm acknowledge support by the Austrian Science Fund (FWF, project P31336). T. Hamm also acknowledges support by the FWF project W1255-N23.
References


