# Petri Net Induced Heuristic Search for Resource Constrained Scheduling

**Ido Lublin, Dor Atzmon, Izack Cohen**

Bar-Ilan University, Israel
{ido.lublin, dor.atzmon, izack.cohen}@biu.ac.il

## Abstract

We formulate the *Resource-Constrained Project Scheduling Problem* (RCPSP) as optimal search over the reachability graph of a *Timed Transition Petri Net with Resources*, using relative-delay tokens so that scheduling decisions correspond to transition firings in the induced state space. We solve the resulting problem with $A^*$ guided by a heuristic that combines Critical Path and resource-based lower bounds, and prove that the heuristic is consistent under our token-based time semantics. Experiments on the PSPLIB benchmarks show that the approach outperforms strong exact Mixed-Integer Linear Programming (MIP) baselines (SCIP, CBC) in success rate and solve time. Per-instance analysis shows that heuristic search and MIP degrade along independent axes, resource tightness for $A^*$ and formulation size for MIP, with resource strength mediating which solver benefits from scale.

## Introduction

The *Resource-Constrained Project Scheduling Problem* (RCPSP) asks how to schedule precedence-constrained activities under limited shared resources so as to minimize completion time (Herroelen, De Reyck, and Demeulemeester 1998; Brucker et al. 1999; Hartmann and Briskorn 2022). As a strongly NP-hard generalization of classical scheduling problems (Blazewicz, Lenstra, and Kan 1983), RCPSP is a fundamental problem at the intersection of scheduling, combinatorial search, and resource-bounded reasoning. The same activity–precedence–resource structure appears in temporal planning and scheduling, robotic and autonomous mission coordination, manufacturing systems, multiprocessor and workflow scheduling, and project execution under limited shared resources. RCPSP also arises as a computational step in a variety of resource-related settings. For example, Cohen (2024) learns project networks from historical project data and uses resource-constrained scheduling procedures to derive schedules and resource allocations. RCPSP has long served as a benchmark for exact and approximate methods, yet no single method dominates across all instance classes.

The leading exact solution approach is *Mixed-Integer Linear Programming* (MIP), typically solved by branch-and-cut (Dorndorf, Pesch, and Phan-Huy 2000; Koné et al. 2011; Artigues et al. 2015). While MIP provides optimality guarantees, it scales poorly, and strong solvers, such as SCIP (Achterberg 2009), often fail to solve benchmark instances within practical time limits. Metaheuristics (Pellerin, Perrier, and Berthaut 2020) improve scalability at the cost of optimality, and hybrid methods seek a middle ground, yet no existing approach reliably achieves both. A natural exact alternative is *heuristic search* (Hart, Nilsson, and Raphael 1968). Bell and Park (1990) applied $A^*$ to RCPSP through conflict sequencing, but the method becomes intractable on highly resource-constrained instances. Zamani and Shue (1998) introduced an activity-dispatch state space and applied $A^*$ via *SLA**. However, this formulation permits the redundant re-expansion of equivalent scheduling states and requires multiple search trials for optimality. We address this inefficiency by proposing a different activity-dispatch state space that inherently captures state equivalencies to enable superior duplicate pruning. More broadly, a recent survey (Huang et al. 2023) highlights the promise of heuristic search over reachability graphs for balancing solution quality and computational effort. Related Petri-net-based ideas have been explored in project and manufacturing scheduling (Mejia et al. 2016; Mejia and Odrey 2005), directed heuristic search over Petri net reachability graphs has been studied in the model checking context (Edelkamp and Jabbar 2006), and simulation has been used to estimate RCPSP project duration (Wu et al. 2009). To the best of our knowledge, no prior work has applied optimal heuristic search to the induced reachability graph of an RCPSP formulation.

This paper develops a new RCPSP formulation as an optimal search over the reachability graph induced by a *Timed Transition Petri Net with Resources* (TTPNR), making local scheduling decisions explicit and representing time uniformly through token delays. This yields an exact search formulation with admissible heuristics, safe duplicate pruning, and efficient treatment of zero-cost transitions. We prove that the combined Critical Path and resource-based heuristic is consistent, and show on standard PSPLIB benchmarks that the resulting approach is competitive with strong exact MIP baselines, including SCIP and CBC.

## RCPSP Problem Definition

RCPSP is characterized by the tuple $(\mathbb{A}, E, R, C, U, \tau)$:

- $\mathbb{A} = \{0, 1, \ldots, n, n+1\}$: set of activities, where 0 and $n+1$ are dummy start and finish.
- $E \subset \mathbb{A} \times \mathbb{A}$: precedence constraints, where $(i, j) \in E$ means $j$ cannot start before $i$ completes. Let $\bullet j = \{i \mid (i, j) \in E\}$ be the set of immediate predecessors of $j$.
- $R = \{r_1, \ldots, r_k\}$: renewable resource types with capacities $C = \{c_1, \ldots, c_k\}$.
- $u_{j,i}$: demand of activity $j$ for resource $r_i$.
- $\tau_j$: duration of activity $j$.

As in standard RCPSP formulations, the dummy activities have zero duration ($\tau_0 = \tau_{n+1} = 0$) and zero resource demand ($u_{0,i} = u_{n+1,i} = 0$ for all $r_i \in R$).

A project *schedule* assigns a start time $S_j \geq 0$ to each activity $j \in \mathbb{A}$. A feasible schedule satisfies two constraints:

1. *Precedence:* An activity cannot start until all of its predecessors complete (for every $(i, j) \in E$, $S_j \geq S_i + \tau_i$).
2. *Resource:* At any time $t$, the total amount of each resource type $r_i \in R$ allocated to ongoing activities cannot exceed its available capacity $c_i \in C$.

The objective is to find a schedule with minimal makespan, that is, the start time of the dummy finish activity: $\min S_{n+1}$. The interaction between precedence and resource constraints creates a highly constrained combinatorial problem, which we later formulate both as a reachability graph and as a MIP. We use the next example throughout the paper.

**Example 1.** *Consider an RCPSP instance with activities $\mathbb{A} = \{0, 1, 2, 3, 4, 5\}$ (denote by $0 \equiv a,\ 1 \equiv b,\ 2 \equiv c,$ etc.). $a$ and $f$ are the dummy start and finish activities. The precedence set is $E = \{(a, b), (a, d), (b, c), (d, e), (c, f), (e, f)\}$, forming two parallel chains $a \to b \to c \to f$ and $a \to d \to e \to f$. There is one type of a renewable resource $r$ with capacity $c_1 = 2$. Each non-dummy activity requires one unit, i.e., $u_{b,1} = u_{c,1} = u_{d,1} = u_{e,1} = 1$, and $u_{a,1} = u_{f,1} = 0$. The durations are $\tau_a = \tau_f = 0, \quad \tau_b = 3, \quad \tau_c = 1, \quad \tau_d = 3, \quad \tau_e = 2$. An optimal schedule has makespan 5 with $S_a = S_b = S_d = 0, \quad S_c = S_e = 3, \quad S_f = 5$. Figure 1(a) summarizes the problem instance parameters.*

## Timed Transition Petri Net with Resources

A *Petri net* (Petri 1966) is a bipartite graph consisting of *places* (denoted by circles) and *transitions* (rectangles) connected by directed arcs. Places hold *tokens*, whose distribution defines the *marking* $M$. Formally, a Petri net is the tuple $(P, T, F, W, M_0)$, where $P$ and $T$ are finite sets of places and transitions, $F \subseteq (P \times T) \cup (T \times P)$ is the arc set, $W : F \to \mathbb{Z}^+$ is a weight function, and $M_0$ is the initial marking. We write $\bullet t = \{p \in P \mid (p, t) \in F\}$ for the input places of transition $t$. A transition $t$ is *enabled* if every place $p \in \bullet t$ holds at least $W(p, t)$ tokens; firing consumes and deposits tokens accordingly, thereby modeling concurrency and synchronization. In our formulation, each reachable marking corresponds to a search state, and the reachable markings define the *reachability graph* that our approach generates and searches incrementally.

To distinguish the original RCPSP formulation from its Petri-net encoding, we use $j \in \mathbb{A}$ for RCPSP activities and $t \in T$ for Petri-net transitions. Below, each transition corresponds to a single activity. We extend the classical Petri net to a TTPNR with time and resource perspectives,

$$\mathcal{N} = (P, T, F, W, M_0, \lambda, \tau_T, R, C),$$

where $\lambda : T \to \mathbb{A}$ maps each Petri-net transition to its corresponding RCPSP activity, $\tau_T : T \to \mathbb{N}_0$ assigns a duration to each transition, and $R, C$ define the resource types and their capacities. Each transition inherits the duration of its associated activity ($\tau_T(t) := \tau_{\lambda(t)}$). A dedicated subset $P_R \subset P$ of resource places enforces capacity constraints: $M_0$ allocates exactly $c_i$ tokens to each resource place. For each transition $t$ and resource type $r_i$, the arcs between $t$ and the corresponding resource place have weight $u_{\lambda(t),i}$. Firing $t$ consumes this many resource tokens and returns them after duration $\tau_T(t)$. The single enabling rule $M(p) \geq W(p, t)$ then guarantees that an activity can start only if both its precedence conditions and resource requirements are satisfied. This unified enabling rule is a key advantage over prior RCPSP search formulations, where precedence and resource constraints require separate conflict-resolution mechanisms outside the search space.

Figure 1(b) illustrates the TTPNR formulation of the RCPSP instance from Example 1 (Figure 1(a)). Here, $a$ and $f$ denote the dummy start and dummy finish activities, respectively. The figure shows the initial marking $M_0$, in which a single token is placed in $p_1$ and two in the resource place $p_r$ (resource type index is omitted for readability). The only enabled transition at $M_0$ is $a$; firing it consumes the token from $p_1$ and produces tokens in $p_2$ and $p_3$, making activities $b$ and $d$ available to compete for the shared resource.

## Modeling RCPSP as a Search Problem

To solve the RCPSP via heuristic search, we induce a reachability graph directly from the TTPNR. In this search graph, each node represents a system state, and each edge corresponds to the commitment to execute a specific project activity by firing a Petri-net transition. A *state* $s \in \mathbb{S}$ is a marking augmented with relative delays on tokens. By encoding time as token delays $\theta$, we avoid a global clock, allowing more states to be recognized as identical and safely pruned. Heuristic consistency then guarantees that the first expansion of any state is optimal. Let $P_C = P \setminus P_R$ denote the control-flow places. In our RCPSP construction, each $p_i \in P_C$ contains at most one token. A state is therefore:

$$s = \big[\, p_i : 1^{\theta_i} \mid p_i \in P_C,\ M_s(p_i) = 1,\ p_{r_1} : R_1, \ldots, p_{r_k} : R_k \,\big].$$

Here, $1^{\theta_i}$ denotes the single control-flow token in place $p_i$, and $\theta_i$ is its residual delay. The explicit multiplicity 1 is included to match the compact state labels used in Figure 1(c).

For each resource place $p_{r_j}$, $R_j = \{\mu_{j,1}^{\theta_{j,1}}, \ldots, \mu_{j,m_j}^{\theta_{j,m_j}}\}$ is a multiset representation of its resource tokens grouped by residual delay. This multiset notation is needed only for resource places, since several interchangeable (same type) resource tokens may occupy the same resource place with different residual delays. Here, $\mu_{j,q}^{\theta_{j,q}}$ denotes $\mu_{j,q}$ interchangeable resource tokens that become available after delay $\theta_{j,q}$.

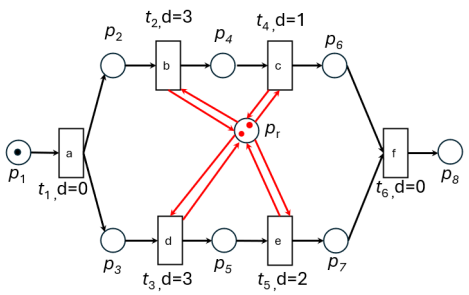
**Problem instance**

| act. | succ. | dur. | $r$ |
|---|---|---|---|
| $a$ | $b, d$ | 0 | 0 |
| $b$ | $c$ | 3 | 1 |
| $c$ | $f$ | 1 | 1 |
| $d$ | $e$ | 3 | 1 |
| $e$ | $f$ | 2 | 1 |
| $f$ | — | 0 | 0 |

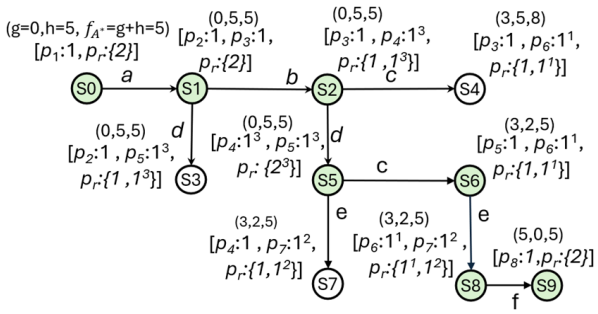
**Capacity**

| $r$ | 2 |
|---|---|

(a)




(b)




(c)

Figure 1: (a) RCPSP instance; (b) TTPNR initial marking; (c) $A^*$ search over the induced reachability graph. Nodes show $(g, h, f)$ and token delays $\theta$, green nodes are expanded.

A token with delay $\theta > 0$ is not yet available, while $\theta = 0$ means immediate availability. We omit zero delays in superscripts: thus $p_i : 1$ denotes one immediately available control-flow token in $p_i$, and $p_i : 1^3$ denotes one control-flow token in $p_i$ available after delay 3. Similarly, $p_{r_j} : \{1, 1^3\}$ denotes one immediately available resource token and one resource token available after delay 3. Only marked control-flow places and resource places with positive token counts are shown.

The *start state* $s_0$ has one token in the input place of the dummy start transition and all resource places at their initial marking; the *goal state* has one token in the output place of the dummy finish transition with resource places restored.

We consider a transition $t \in T$ *enabled* if its input places contain sufficient tokens, irrespective of their current relative delays. Firing $t$ commits the corresponding activity $\lambda(t)$ and proceeds as follows. (1) *Time jump:* From each input place $p \in {\bullet t}$, consume the required number of tokens, choosing those with smallest relative delay in that place. The induced time jump is $\Delta t = \max(\theta_{\text{consumed}})$ over all consumed tokens, because all consumed tokens must be available before the activity can start, and the latest available consumed token determines the earliest feasible start time. (2) *Delay and token generation:* All remaining token delays are reduced by $\Delta t$: $\theta^{\text{new}} = \max(0, \theta - \Delta t)$. Newly produced tokens are assigned the delay $\tau_T(t) = \tau_{\lambda(t)}$. Also, each node maintains the path cost $g(s') = g(s) + \Delta t$ for use by $A^*$.

Thus, firing $t$ schedules activity $\lambda(t)$ at absolute time $g(s) + \Delta t$, namely at the earliest time allowed by the tokens consumed from the partial schedule encoded by $s$.

**Proposition 1.** *For every RCPSP instance, the minimum-cost path from the initial state to the goal state in the TTPNR-induced search graph has cost equal to the optimal RCPSP makespan.*

*Proof.* We prove the proposition by showing both directions: first, that any goal-reaching TTPNR path induces a feasible RCPSP schedule, and then that an optimal RCPSP schedule is realized by a goal-reaching TTPNR path.

By construction, activities, precedence relations, and renewable resources correspond to transitions, control-flow places, and resource places, respectively.

A transition can be selected only when the required control-flow and resource tokens exist; the relative delays of these tokens determine the necessary time jump before the corresponding activity starts.

Hence, any path from the initial state to the goal state induces a feasible RCPSP schedule: precedence is respected by the control-flow tokens, resource capacities are respected by the resource tokens, and the accumulated path cost $g$ equals the elapsed project time. When the dummy finish transition fires, $g$ is the makespan of the induced schedule.

Conversely, since RCPSP makespan minimization has at least one optimal active schedule (Kolisch 1996), it suffices to show that the TTPNR-induced graph contains a goal-reaching path that realizes one such schedule.

By the standard serial schedule-generation argument, such an optimal active schedule can be obtained by following a precedence-feasible activity list and scheduling each activity at its earliest feasible start time. Note that the precedence-feasible activity list for an optimal active schedule can be realized by the TTPNR-induced graph, with ties between activities that start at the same time broken in any precedence-respecting order. Immediately before each selected activity is fired, the required control-flow and resource tokens are present, possibly with positive residual delays. The induced time jump advances the state to the earliest time at which all consumed tokens are available. Since renewable resource units of the same type are interchangeable, consuming the smallest-delay resource tokens is without loss of generality. The relative-delay semantics therefore schedules the selected activity at its earliest feasible start time with respect to the partial schedule constructed so far, and then assigns the activity duration as the delay on the produced control-flow and resource tokens. Applying this argument inductively over the activity list yields a path that reaches the goal marking with cost equal to the optimal makespan.

Since every goal-reaching path induces a feasible RCPSP schedule, no such path can have cost smaller than the optimal makespan. The path constructed above reaches the goal with cost equal to the optimal makespan. Therefore, the minimum-cost path from the initial state to the goal state has cost exactly equal to the optimal RCPSP makespan. □

Other firing orders may generate non-active feasible schedules. This does not affect the result, because the proof only requires that an optimal path exists in the induced graph.

## Heuristics

We use two lower bounds on the remaining makespan from a state $s$: the critical-path bound $h_{CP}(s)$, obtained by relaxing resource capacities, and the resource-load bound $h_{res}(s)$, obtained by relaxing precedence constraints. Their combination is $h_{\max}(s) = \max(h_{CP}(s),\, h_{res}(s))$.

**Critical-Path Heuristic** $h_{CP}(s)$ is the length of the longest *residual* precedence path to the dummy finish activity $n{+}1$, computed by propagating *residual* earliest finish times, $REF$, in topological order:

$$REF_j = \begin{cases} 0 & \text{if } j \text{ is completed,} \\ \theta_j & \text{if } j \text{ is currently executing,} \\ \max\limits_{i \in \bullet j} REF_i + \tau_j & \text{if } j \text{ has not yet started,} \end{cases}$$

initialized with $REF_0 = 0$, and $h_{CP}(s) = REF_{n+1}$.

**Proposition 2.** *$h_{CP}$ is admissible and consistent.*

*Proof. Admissibility.* Removing resource constraints cannot extend the remaining completion time, so $h_{CP}(s) \le h^*(s)$.

*Consistency.* Consider a successor $s'$ of $s$ with time increment $\delta(s, s') := g(s') - g(s)$. For every activity $j$, its residual duration decreases by at most $\delta(s, s')$: executing activities lose at most $\delta(s, s')$, unstarted activities are unchanged, and a newly started activity $\lambda(t)$ receives residual delay $\tau_{\lambda(t)}$ in $s'$, identical to its value before firing. Since precedence enforces sequential execution along any single path, at most one activity on the longest path $P^*$ is executing in $s$, so the total residual length of $P^*$ decreases by at most $\delta(s, s')$. Since $h_{CP}(s')$ is the maximum over all paths,

$$h_{CP}(s') \ge h_{CP}(s) - \delta(s, s'),$$

which rearranges to $h_{CP}(s) \le c(s, s') + h_{CP}(s')$. □

**Resource-Based Heuristic** $h_{res}(s)$ is the minimum time to clear the remaining workload on the most heavily loaded resource, ignoring precedence constraints. Let $\mathrm{active}(s)$ and $\mathrm{unstarted}(s)$ denote the currently executing and not-yet-started activities, respectively.

$$h_{res}(s) = \max_{r \in R} \frac{\sum\limits_{a \in \mathrm{active}(s)} \theta_a\, u_{a,r} + \sum\limits_{a \in \mathrm{unstarted}(s)} \tau_a\, u_{a,r}}{c_r}.$$

**Proposition 3.** *$h_{res}$ is admissible and consistent.*

*Proof. Admissibility.* Dropping precedence constraints can only reduce the remaining makespan, so $h_{res}(s) \le h^*(s)$.

*Consistency.* As for $h_{CP}$: firing $t$ advances the clock by $\delta(s, s')$, reducing the residual workload numerator by at most $\delta(s, s') \cdot c_r$ per resource (executing activities lose at most $\delta(s, s') \cdot u_{a,r} \le \delta(s, s') \cdot c_r$, unstarted activities are unchanged, and $\lambda(t)$ satisfies $\theta_{\lambda(t)} = \tau_{\lambda(t)}$ immediately after firing). Dividing by $c_r$ and taking the maximum over resources yields $h_{res}(s) \le c(s, s') + h_{res}(s')$. □

**Proposition 4.** *A combined heuristic*

$$h_{\max}(s) = \max(h_{CP}(s), h_{res}(s))$$

*is admissible and consistent.*

*Proof.* The max over admissible and consistent heuristics is admissible and consistent (Russell and Norvig 2020). □

Consistency guarantees that $A^*$ remains optimal while eliminating the need to reopen expanded states (Hart, Nilsson, and Raphael 1968; Dechter and Pearl 1985). Our formulation also enables an effective caching optimization: whenever $\delta(s, s') = 0$, both lower bounds are preserved, so $h_{\max}(s') = h_{\max}(s)$. Thus, zero-cost successors can reuse the parent's cached heuristic value, without recomputation.

The relative-delay token representation, coupled with a consistent heuristic, naturally induces an effective duplicate-detection mechanism. Two states with identical token distributions will produce identical subtrees going forward; the state representation therefore identifies them as the same state regardless of their absolute start times. Since $h_{\max}$ is consistent, $A^*$ guarantees that the first time a state is expanded, it is via the lowest $g$ path, so duplicates are safely discarded without risk of losing an optimal solution.

In the resulting $A^*$ search, we prioritize nodes with minimal $f_{A^*}(s) = g(s) + h_{\max}(s)$. Ties in this value are broken by preferring higher $g$-cost, then more finished activities, then more active activities, then FIFO, driving the search depth-first through equal-$f_{A^*}$ plateaus.

Figure 1(c) illustrates the $A^*$ guided search over the induced reachability graph of Example 1.

# MIP Formulation

While TTPNR formulates RCPSP as a reachability problem amenable to heuristic search, MIP formulates the same problem as a mathematical optimization problem solved via branch-and-cut. The two frameworks are alternative mathematical formulations of the same underlying problem, inducing fundamentally different solution mechanisms.

Since MIP-based branch-and-cut is the dominant exact RCPSP approach (Dorndorf, Pesch, and Phan-Huy 2000), we compare against two strong solvers, SCIP (Achterberg 2009) and CBC (Forrest and Lougee-Heimer 2005), using the standard time-indexed formulation (Artigues et al. 2015): a binary variable $x_{jt} \in \{0, 1\}$ indicates whether activity $j$ starts at time $t$. The full encoding is:

$$\min \quad \sum_{t \in H} t \cdot x_{n+1,t} \tag{1}$$

$$\text{s.t.} \quad \sum_{t \in H} x_{jt} = 1 \quad \forall j \in \mathbb{A} \tag{2}$$

$$S_i + \tau_i \le S_j \quad \forall (i, j) \in E \tag{3}$$

$$\sum_{j \in \mathbb{A}} u_{j,r} \sum_{q = t - \tau_j + 1}^{t} x_{jq} \le c_r \quad \forall r \in R,\ \forall t \in H \tag{4}$$

Constraints (2)–(4) ensure activities start exactly once, enforce precedence and resource capacity, with $S_j = \sum_{t \in H} t\, x_{jt}$ and $H = \{0, \ldots, T\}$, where $T = \sum_{j \in \mathbb{A}} \tau_j$.

| Metric / Parameter | J30 | | | J60 | | | J90 | | |
|---|---|---|---|---|---|---|---|---|---|
| | TTPNR | CBC | SCIP | TTPNR | CBC | SCIP | TTPNR | CBC | SCIP |
| *Overall Performance* | | | | | | | | | |
| **Success Rate (%)** | *(+27.5)* **98.96** | 47.29 | 71.46 | *(+19.6)* **47.71** | 28.13 | 8.75 | *(+23.1)* **36.04** | 12.92 | 0.21 |
| **Avg. Solved Time (s)** | *(-78.8%)* **7.97** | 42.14 | 37.65 | *(-77.5%)* **19.33** | 85.96 | 186.42 | *(-96.2%)* **5.97** | 197.37 | 155.97 |
| *Success Rate (%) by Resource Strength (RS)* | | | | | | | | | |
| **RS = 1.0** | **100.00** | 95.00 | **100.00** | **94.17** | 56.67 | 27.50 | **87.50** | 30.83 | 0.83 |
| **RS = 0.7** | **98.33** | 60.83 | 95.83 | **71.67** | 46.67 | 5.00 | **43.33** | 18.33 | 0.00 |
| **RS = 0.5** | **99.17** | 24.17 | 68.33 | **22.50** | 9.17 | 1.67 | **13.33** | 2.50 | 0.00 |
| **RS = 0.2** | **98.33** | 9.17 | 22.50 | **2.50** | 0.00 | 0.83 | 0.00 | 0.00 | 0.00 |
| *Success Rate (%) by Resource Factor (RF)* | | | | | | | | | |
| **RF = 1.0** | **100.00** | 38.33 | 51.67 | **49.17** | 35.83 | 3.33 | **40.00** | 15.00 | 0.83 |
| **RF = 0.75** | **100.00** | 38.33 | 65.00 | **49.17** | 40.83 | 5.83 | **34.17** | 15.83 | 0.00 |
| **RF = 0.50** | **98.33** | 44.17 | 72.50 | **50.83** | 30.00 | 5.83 | **33.33** | 19.17 | 0.00 |
| **RF = 0.25** | **97.50** | 68.33 | **97.50** | **41.67** | 5.83 | 20.00 | **36.67** | 1.67 | 0.00 |

Table 1: Overall Success Rate and Average Time, alongside breakdowns by Resource Strength (RS) and Resource Factor (RF).

## Experiments and Discussion

We evaluated $A^*$ (denoted TTPNR), SCIP, and CBC on the standard public RCPSP benchmark suite PSPLIB (Kolisch and Sprecher 1997), using the J30, J60, and J90 sets (30, 60, and 90 activities; 480 instances per set). Each instance has four renewable resource types, and the instances in each set are organized into 48 groups of 10 with shared generation parameters. These parameters (discussed later) control structural properties and resource tightness, yielding a diverse collection of challenging RCPSP instances. Evaluations were run sequentially on a single-core Intel Xeon Gold 6248R CPU @ 3.00GHz with a 5-minute timeout. To ensure a rigorous comparison of the underlying search engines and branching mechanics, the MIP solvers and TTPNR were executed without preprocessing. TTPNR was implemented in C++ and is publicly available on GitHub.[1]

Table 1 summarizes success rate, average runtime over solved instances, and performance breakdowns by problem features. TTPNR outperforms both MIP solvers in success rate and is substantially faster across all datasets.

| | J30 | J60 | J90 |
|---|---|---|---|
| Instances used | 5 | 221 | 287 |
| Avg. gap (%) | 3.74 | 5.09 | 2.71 |

Table 2: Lower-bound gap by TTPNR (unsolved instances).

Table 2 reports the average lower bound gap, in percent, computed only over instances that were not solved by TTPNR within the time limit and for which a reference value is available (5, 221, 287 instances for J30, J60, and J90):

$$100 \cdot \frac{LB_{known} - LB_{TTPNR}}{LB_{known}}.$$

For instances solved by TTPNR, the optimal value was proven and the corresponding optimality gap is zero. All 480 instances in the J30 dataset have known optimal values. For the J60 and J90 datasets, published lower bounds are available for 410 and 445 out of 480 instances, respectively.

[1]https://github.com/idoLublin/Petri-Net-Induced-Heuristic-Search-for-Resource-Constrained-Scheduling-SoCS-2026

## Impact of Problem Features on TTPNR

*Network Complexity (NC)* measures how strongly precedence constraints restrict activity order, counting only non-redundant arcs (redundant arcs are implied by transitivity). Higher NC makes the schedule more sequential, reducing the number of simultaneously enabled activities, reducing the branching factor. From our experiments, NC proved to be the least impactful parameter for both MIP solvers and TTPNR, and is thus omitted from Table 1.

*Resource Strength (RS)* measures how restrictive the resource capacities are. At RS = 1, resources are abundant enough that competition for resources is minimal, while RS $= 0$ implies that capacities are near their minimum feasible levels and competition for resources is maximal. When RS is low, many delays are caused by resource conflicts rather than precedence, so the true remaining makespan can be much larger than the precedence-only lower bound. This makes $h_{CP}$, and therefore $h_{\max}$, less informative for search, and reducing success rates (RS section of Table 1).

*Resource Factor (RF)* measures the average proportion of resource types required per activity. While SCIP generally degrades as RF increases, CBC exhibits mixed sensitivity. TTPNR demonstrates robustness to RF variations. In TTPNR multi-resource requirements involve transitions concurrently consuming tokens from multiple places, preventing branching factor inflation and ensuring stable success rates for any RF (RF section of Table 1).

## Conclusion

We formulated RCPSP as optimal search on the reachability graph of a TTPNR with relative-delay tokens, solved via $A^*$. Empirically, heuristic search and MIP degrade along independent axes—resource tightness for $A^*$, formulation size for MIP—with resource strength mediating which solver benefits from scale. Future work includes stronger resource-aware heuristics for low-resource-strength instances, algorithm selection across paradigms, and bidirectional search such as BAE* (Sadhukhan 2013), which fit our state representation. The principle extends to variants like multi-mode RCPSP (Balouka, Cohen, and Shtub 2015), and parallel machine scheduling (Cohen, Cohen, and Zaks 2024).

## Acknowledgements

This research was supported by the Israel Science Foundation (ISF), grant no. 353/24 and grant no. 1511/25, by the German Research Foundation under grant reference KR 4926/4-1, AOBJ 696883, and by Israel's Ministry of Innovation, Science and Technology (MOST) grant #6908 (Czech-Israeli cooperative scientific research).

## References

Achterberg, T. 2009. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1: 1–41.

Artigues, C.; Koné, O.; Lopez, P.; and Mongeau, M. 2015. Mixed-integer linear programming formulations. *Handbook on project management and scheduling vol. 1*, 17–41.

Balouka, N.; Cohen, I.; and Shtub, A. 2015. Extending the multimode resource-constrained project scheduling problem by including value considerations. *IEEE Transactions on Engineering Management*, 63(1): 4–15.

Bell, C. E.; and Park, K. 1990. Solving resource constrained project scheduling problems by A* search. *Naval Research Logistics (NRL)*, 37(1): 61–84.

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

Brucker, P.; Drexl, A.; Möhring, R.; Neumann, K.; and Pesch, E. 1999. Resource-constrained project scheduling: Notation, classification, models, and methods. *European journal of operational research*, 112(1): 3–41.

Cohen, I. 2024. Data-driven project planning: An integrated network learning, process mining, and constraint relaxation approach in favor of scheduling recurring projects. *IEEE Transactions on Engineering Management*, 71: 7719–7729.

Cohen, I. R.; Cohen, I.; and Zaks, I. 2024. A theoretical and empirical study of job scheduling in cloud computing environments: the weighted completion time minimization problem with capacitated parallel machines. *Annals of Operations Research*, 338(1): 429–452.

Dechter, R.; and Pearl, J. 1985. Generalized best-first search strategies and the optimality of A*. *J. ACM*, 32(3): 505–536.

Dorndorf, U.; Pesch, E.; and Phan-Huy, T. 2000. A branch-and-bound algorithm for the resource-constrained project scheduling problem. *Mathematical Methods of Operations Research*, 52: 413–439.

Edelkamp, S.; and Jabbar, S. 2006. Action Planning for Directed Model Checking of Petri Nets. *Electronic Notes in Theoretical Computer Science*, 149(2): 3–18.

Forrest, J.; and Lougee-Heimer, R. 2005. CBC User Guide. Technical report, COIN-OR Foundation.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

Hartmann, S.; and Briskorn, D. 2022. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research*, 297(1): 1–14.

Herroelen, W.; De Reyck, B.; and Demeulemeester, E. 1998. Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25(4): 279–302.

Huang, B.; Zhou, M.; Lu, X. S.; and Abusorrah, A. 2023. Scheduling of resource allocation systems with timed Petri nets: A survey. *ACM Computing Surveys*, 55(11): 1–27.

Kolisch, R. 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *European Journal of Operational Research*, 90(2): 320–333.

Kolisch, R.; and Sprecher, A. 1997. PSPLIB-a project scheduling problem library: OR software-ORSEP operations research software exchange program. *European Journal of Operational Research*, 96(1): 205–216.

Koné, O.; Artigues, C.; Lopez, P.; and Mongeau, M. 2011. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1): 3–13.

Mejia, G.; Nino, K.; Montoya, C.; Sanchez, M. A.; Palacios, J.; and Amodeo, L. 2016. A Petri Net-based framework for realistic project management and scheduling: An application in animation and videogames. *Computers & Operations Research*, 66: 190–198.

Mejia, G.; and Odrey, N. G. 2005. An approach using Petri nets and improved heuristic search for manufacturing system scheduling. *Journal of Manufacturing Systems*, 24(2): 79–92.

Pellerin, R.; Perrier, N.; and Berthaut, F. 2020. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 280(2): 395–416.

Petri, C. A. 1966. Communication with automata. *Technical Report No. RADC-TR-65-377*.

Russell, S.; and Norvig, P. 2020. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition.

Sadhukhan, S. K. 2013. Bidirectional heuristic search based on error estimate. *CSI Journal of Computing*, 2(1-2): S1:57–S1:64.

Wu, Y.; Zhuang, X.-c.; Song, G.-h.; Xu, X.-d.; and Li, C.-x. 2009. Solving resource-constrained multiple project scheduling problem using timed colored Petri nets. *Journal of Shanghai Jiaotong University (Science)*, 14: 713–719.

Zamani, R.; and Shue, L.-Y. 1998. Solving project scheduling problems with a heuristic learning algorithm. *Journal of the Operational Research Society*, 49(7): 709–716.