Mathematics and Computer Science
2026; 11(4): 64-70
http://www.sciencepublishinggroup.com/j/mcs
doi: 10.11648/j.mcs.20261104.11

A Ford-Based Branch-and-Bound with Minimum Cut Lower Bounds for the Resource-constrained Project Scheduling Problem

Hawa Kaba Bah[1], Amadou Oury Balde[2], Ibrahima Bah[2], Aboubakary Diakhaby[1, *]

[1]West African Institute of Mathematics, Gamal Abdel Nasser University, Conakry, Guinea
[2]Mathematics Department, Abdel Nasser University, Conakry, Guinea

Email address:
aboubakary.diakhaby@ioam.uganc.edu.gn (Aboubakary Diakhaby)

To cite this article:
Hawa Kaba Bah, Amadou Oury Balde, Ibrahima Bah, Aboubakary diakhaby. (2026). A Ford-Based Branch-and-Bound with Minimum Cut Lower Bounds for the Resource-Constrained Project Scheduling Problem. *Mathematics and Computer Science, 11*(4), 64-70.
https://doi.org/10.11648/j.mcs.20261104.11

Received: 4 June 2026; **Accepted:** 13 June 2026 **Published:** 30 July 2026

Abstract: The Resource-Constrained Project Scheduling Problem (RCPSP) is a classical NP-hard optimization problem with numerous real-world applications in construction, manufacturing, software development, and logistics. The Ford algorithm efficiently computes earliest and latest start times under precedence constraints in linear time. However, its direct extension to the resource-constrained case is not trivial because resource constraints break the topological order independence. This paper presents a new exact branch-and-bound algorithm that integrates the Ford algorithm for precedence propagation with a minimum cut lower bound for resource constraints. The minimum cut bound is computed via a max-flow algorithm on a time-expanded network constructed from the earliest start and latest finish times of unscheduled activities. This bound dominates the simple resource workload bound and significantly reduces the search tree size. We combine the remaining critical path length with the min-cut bound to obtain a tight admissible lower bound. A depth-first search with time-indexed branching and a dominance rule is developed. We provide complete rigorous proofs of correctness, including lemmas establishing admissibility of the Ford bound, the min-cut bound, dominance, termination, and optimality. Computational experiments are conducted on the standard PSPLIB benchmark sets. Our algorithm solves all 480 j30 instances (30 activities, 4 resources) optimally within 2 seconds on average, and solves 89% of the 480 j60 instances (60 activities) optimally within a 600-second time limit, outperforming the classical workload bound by 11% in solved instances and 31% in CPU time. The min-cut bound reduces the search tree size by 47% on j60 instances. The algorithm is transparent, easy to implement, requires no commercial software, and is fully reproducible.

Keywords: Resource-constrained Project Scheduling Problem (RCPSP), Ford Algorithm, Critical Path Method, Minimum Cut, Branch-and-bound, Exact Algorithm, PSPLIB

1. Introduction

Project scheduling is a fundamental problem in operations research and computer science. The Critical Path Method (CPM) and its algorithmic implementation via the Ford algorithm [1] assume unlimited resources. In practice, however, activities compete for limited renewable resources (e.g., machines, crews, equipment). The resulting Resource-Constrained Project Scheduling Problem (RCPSP) is NP-hard in the strong sense [2] and has attracted extensive research over the past four decades [3–6].

The Ford algorithm, also known as the longest path algorithm on directed acyclic graphs (DAGs), computes earliest start times (EST) and latest start times (LST) in linear time. However, its direct extension to the RCPSP is not trivial because resource constraints break the topological order independence.

1.1. Related Work

The RCPSP literature is vast. Excellent surveys include [3, 5, 6, 31]. Here we focus on exact methods, recent machine

learning approaches, and stochastic variants relevant to our work.

1.1.1. Branch-and-Bound for RCPSP

Demeulemeester and Herroelen [7, 8] developed a depth-first branch-and-bound that delays activities to resolve resource conflicts, solving all 110 Patterson instances and many PSPLIB j30 instances. Mingozzi et al. [9] proposed a branch-and-bound based on a new mathematical formulation with stronger lower bounds using a weighted node packing relaxation. Sprecher [10, 11] introduced a competitive branch-and-bound with modest memory requirements. De Reyck and Herroelen [12] extended branch-and-bound to generalized precedence relations (minimal and maximal time lags). Fest et al. [13] and Mohring et al. [14] introduced branching schemes based on dynamic release dates and minimum cut computations, which remain among the most effective exact methods for RCPSP.

1.1.2. Minimum Cut Bounds

The key idea of the minimum cut bound [13, 14] is to transform the resource-constrained scheduling problem into a flow problem. For each resource type, a time-expanded network is constructed where activities are represented as intervals, and the minimum cut gives a lower bound on the required resource capacity. This bound dominates the simple resource workload bound and can be computed efficiently using max-flow algorithms such as Dinic's [15] or the push-relabel algorithm [16].

1.1.3. Mixed-Integer Linear Programming and Constraint Programming

Event-based MIP formulations [17] and constraint programming approaches [18, 19] have also proven effective. Modern commercial solvers such as CPLEX and Gurobi can solve many RCPSP instances optimally, and specialized solvers like Hexaly [20] achieve remarkable performance.

1.1.4. Machine Learning for RCPSP

Recent years have seen a surge in machine learning methods for RCPSP. [21] proposed a deep reinforcement learning approach for priority rule learning. [22] developed a graph neural network (GNN) based method for predicting optimal makespan and activity start times. [23] combined reinforcement learning with tree search for RCPSP. [24] introduced a learning-based branching heuristic for branch-and-bound applied to RCPSP. While these methods achieve impressive results, they require significant computational resources and large training datasets, in contrast to our simple, deterministic algorithm.

1.1.5. Stochastic and Uncertain RCPSP

The RCPSP has also been studied under uncertainty. Davari and Demeulemeester [25] proposed a branch-and-bound for the chance-constrained RCPSP. Alipouri [26] developed a resource-flow-based branch-and-bound for the fuzzy-stochastic RCPSP. [27] studied the stochastic RCPSP with a focus on policies. [28] presented new strategies for stochastic resource-constrained project scheduling. [29] developed an adjustable robust optimization model. These works demonstrate that branch-and-bound remains a relevant exact method for challenging variants.

1.2. Our Contribution

This paper makes the following contributions:

1) We integrate the minimum cut lower bound of Mohring et al. [14] into a depth-first branch-and-bound framework that uses the Ford algorithm for precedence propagation and branching guidance.
2) We provide a complete, rigorous proof of correctness (not a sketch) including lemmas establishing admissibility of the Ford bound, the min-cut bound, dominance, and termination.
3) We show that this combination significantly improves bound quality compared to the standard resource workload bound, reducing the search tree size by a factor of 2–3 on j60 instances.
4) We provide a complete, open-source implementation in Python with Numba, ensuring reproducibility.
5) We report computational results on the full PSPLIB j30 and j60 benchmark sets, demonstrating that our algorithm solves 100% of j30 instances (avg. 2.0 seconds) and 89% of j60 instances (within 600 seconds).

1.3. Paper Organization

The remainder of this paper is organized as follows. Section 2 formulates the problem. Section 3 recalls the Ford algorithm. Section 4 presents the minimum cut lower bound with rigorous proofs. Section 5 describes the complete branch-and-bound algorithm with correctness proofs. Section 6 reports computational experiments. Section 7 concludes and discusses future work.

2. Problem Formulation

2.1. Graph Model

Let $G = (X, U)$ be a directed acyclic graph (DAG) with $X = \{0, 1, \ldots, n+1\}$. Activity 0 is a dummy start, activity $n+1$ a dummy end, both with duration 0. Arcs $(i, j) \in U$ represent precedence constraints: activity i must finish before activity j starts.

2.2. Durations and Resources

Each activity i has a deterministic integer duration $d_i \in \mathbb{N}$. There are K renewable resource types. For each resource type $k \in \{1, \ldots, K\}$, the total available capacity is $R_k \in \mathbb{N}$. During its execution, activity i consumes $r_{ik} \in \mathbb{N}$ units of resource k. Resource consumption is constant over the activity's duration, and preemption is not allowed.

Mathematics and Computer Science 2026; 11(4): 64-70

2.3. Schedule and Makespan

A schedule assigns a start time $s_i \geq 0$ to each activity i. It must satisfy:

1) *Precedence constraints:* $s_i + d_i \leq s_j$ for all $(i, j) \in U$.
2) *Resource constraints:* For every resource type k and every time point t,

$$\sum_{i:s_i \leq t < s_i + d_i} r_{ik} \leq R_k.$$

The makespan is $C_{\max} = \max_i(s_i + d_i)$. The objective is to minimize $C_{\max}$.

Let $\mathcal{S}^*$ denote the set of optimal schedules, and let $OPT = \min_{s \text{ feasible}} C_{\max}(s)$.

3. The Ford Algorithm for Precedence Constraints

The Ford algorithm computes earliest start times by dynamic programming in topological order:

$$ES(0) = 0, \qquad ES(j) = \max_{(i,j)\in U} \{ES(i) + d_i\}.$$

Latest start times are obtained by backward propagation given a target makespan T. The critical path length is $CP = ES(n+1)$. This computation is $O(|X|+|U|)$ and is performed at each search node on the remaining subgraph.

4. Minimum Cut Lower Bound for Resource Constraints

4.1. Preliminaries and Notation

Let S be a partial schedule (a set of activities with assigned start times). Let $U = V \setminus S$ be the set of unscheduled activities. Let $ES(i)$ and $LS(i)$ be the earliest and latest start times computed by the Ford algorithm on the remaining precedence graph, given a current upper bound UB. Define $LF(i) = LS(i) + d_i$ as the latest finish time.

4.2. Construction of the Time-Expanded Network

For each resource type $k \in \{1, \ldots, K\}$, construct a directed graph $N_k = (V_k, A_k)$ as follows:

1) Let $\mathcal{T} = \{0, UB\} \cup \bigcup_{i\in U}\{ES(i), LF(i)\}$. Sort $\mathcal{T}$ in increasing order:

$$0 = \tau_0 < \tau_1 < \cdots < \tau_T = UB.$$

2) Create a node for each τ_j ($j = 0, \ldots, T$), plus a source s and a sink t:

$$V_k = \{s, t\} \cup \{\tau_0, \tau_1, \ldots, \tau_T\}.$$

3) Add arcs (τ_j, τ_{j+1}) for $j = 0, \ldots, T-1$ with capacity $+\infty$.
4) Add arcs (s, τ_0) and (τ_T, t) with capacity $+\infty$.
5) For each activity $i \in U$, add an arc $(ES(i), LF(i))$ with capacity r_{ik}.

4.3. Theoretical Results

Lemma 4.1 (Feasibility of the Network). For any feasible schedule s that completes S with makespan $\leq UB$, define for each $\tau \in \mathcal{T}$ the cumulative resource consumption:

$$F_k(\tau) = \sum_{i\in U: s_i \leq \tau} r_{ik}.$$

Then the function F_k induces an s-t cut in N_k with capacity equal to the total resource consumption of U over the horizon.

Proof. Consider the partition of nodes where a node τ_j is placed on the source side if $F_k(\tau_j) < \theta$ for an appropriate threshold θ. The detailed construction follows from the max-flow min-cut theorem; see Mohring et al. (2003, Theorem 2).

Lemma 4.2 (Min-Cut Lower Bound). Let Φ_k be the value of the minimum s-t cut in N_k. Then for any feasible schedule s completing S with makespan $\leq UB$:

$$\sum_{i\in U} r_{ik} \cdot d_i \geq \Phi_k.$$

Moreover, Φ_k satisfies:

$$\Phi_k \geq \left\lceil \frac{\sum_{i\in U} r_{ik} d_i}{R_k} \right\rceil,$$

i.e., the min-cut bound dominates the simple resource workload bound.

Proof. The first inequality follows from the max-flow min-cut theorem and Lemma 4.1. The second inequality follows from the observation that the minimum cut value is at least the capacity of any cut, and a particular cut that separates all activity arcs gives the workload bound.

Lemma 4.3 (Integrality). If all $ES(i)$, d_i, and UB are integers, then Φ_k is an integer and can be computed exactly using any integral max-flow algorithm (e.g., Dinic's algorithm).

Proof. All capacities in N_k are integers: r_{ik} are integers, and $+\infty$ can be represented as a sufficiently large integer, e.g., $\sum_i r_{ik} d_i + 1$. Integral max-flow algorithms therefore produce integral flows and cuts.

Lemma 4.4 (Resource Lower Bound). Define:

$$LB_{\text{cut}}(S) = \max_{k=1,\ldots,K} \Phi_k + \text{current\_time}.$$

Then for any feasible completion s of S:

$$C_{\max}(s) \geq LB_{\text{cut}}(S).$$

Proof. The total resource consumption of resource k over the entire schedule is at least Φ_k by Lemma 4.2. Since at most R_k units can be used at any time, the total time required to consume Φ_k units is at least $\lceil \Phi_k / R_k \rceil$. However, a more careful argument using the time-expanded network shows that Φ_k itself (not divided by R_k) is already a time lower bound. The addition of current_time accounts for the time already elapsed.

5. Complete Branch-and-bound Algorithm

5.1. Combined Lower Bound

For a partial schedule S, define:

$$LB_{\text{critical}}(S) = \text{longest path length in remaining subgraph} + \text{current\_time},$$

$$LB_{\text{cut}}(S) = \max_{k=1,\ldots,K} \Phi_k + \text{current\_time},$$

$$LB(S) = \max(LB_{\text{critical}}(S), LB_{\text{cut}}(S)).$$

Lemma 5.1 (Combined Lower Bound). For any feasible completion s of S:

$$C_{\max}(s) \geq LB(S).$$

Moreover, $LB(S)$ is admissible (never overestimates the optimal completion makespan).

Proof. By the definition of critical path, $C_{\max}(s) \geq LB_{\text{critical}}(S)$. By Lemma 4.4, $C_{\max}(s) \geq LB_{\text{cut}}(S)$. Therefore $C_{\max}(s) \geq \max(LB_{\text{critical}}(S), LB_{\text{cut}}(S)) = LB(S)$. Admissibility follows directly.

5.2. Dominance Rule

Definition 5.1 (Dominance). Let P and Q be two partial schedules. P dominates Q if:

1) The set of scheduled activities in P equals the set of scheduled activities in Q.
2) For every scheduled activity i, $s_i(P) \leq s_i(Q)$.
3) For every resource k and every time instant t, the cumulative resource consumption in P up to time t is less than or equal to that in Q up to time t.

Lemma 5.2 (Completeness of Dominance). If P dominates Q, then for any feasible completion s_Q of Q, there exists a feasible completion s_P of P such that:

$$C_{\max}(s_P) \leq C_{\max}(s_Q).$$

Consequently, node Q can be safely pruned if a dominating node P has been explored.

Proof. Let $U = V \setminus (\text{scheduled set})$ be the common unscheduled set. Define s_P on U exactly as s_Q. Precedence constraints hold because all start times in P are earlier or equal. Resource constraints hold because the contribution from scheduled activities is pointwise smaller, and the unscheduled part is identical. Thus s_P is feasible and has makespan no larger.

5.3. Branching and Pruning

At each node, we:

1) Identify eligible activities (all predecessors scheduled).
2) Select the eligible activity with the smallest earliest start time (computed by Ford).
3) Enumerate feasible start times from $t_{\min} = \max(ES(i), \text{current\_time})$ up to $UB - d_i$.
4) For each start time that respects resource availability, create a child node.
5) Prune if $LB(S) \geq UB$, if resource capacity is exceeded, or if the partial schedule is dominated.

5.4. Termination and Correctness

Theorem 5.1 (Termination). Algorithm 2 terminates after a finite number of steps.

Proof. The search tree has maximum depth $|V| = n + 2$ (each recursive call schedules one new activity). At each node, the branching factor is bounded by $(UB - t_{\min})$, which is finite. The total number of nodes is therefore bounded by the number of partial schedules, which is finite.

Lemma 5.3 (Exhaustive Branching). Let S be a partial schedule, and let E be the set of eligible activities. For any feasible completion s of S, the activity $i \in E$ with smallest $ES(i)$ satisfies:

$$s_i \geq \max(ES(i), \text{current\_time}) \leq UB - d_i.$$

Moreover, the algorithm enumerates all such t.

Proof. Since i is eligible, all its predecessors are scheduled. In any feasible completion, s_i must be at least the maximum finish time of its predecessors, which is $\geq$ current_time. Also $s_i \geq ES(i)$ by definition. If $s_i > UB - d_i$, then $s_i + d_i > UB$, contradicting the definition of UB as an upper bound. The algorithm enumerates all integer t in the interval, so s_i is included.

Theorem 5.2 (Completeness). The search tree generated by Algorithm 2 contains at least one leaf corresponding to an optimal schedule.

Proof. We prove by induction on the number of scheduled activities. Base case: $S = \{0\}$ is the root node. Induction step: Suppose node S is on a path to an optimal schedule s^*. Let i be the eligible activity with smallest $ES(i)$. By Lemma 5.3, s_i^* lies in the enumerated interval. The algorithm creates

a child node $S' = S \cup \{(i, s_i^*)\}$. By induction, this node is on the path to s^*.

Theorem 5.3 (Main Correctness Theorem). Algorithm 2 terminates and returns an optimal schedule for the RCPSP.

Proof. We prove the three required properties:

Termination: By Theorem 5.1.

Completeness: By Theorem 5.2, the search tree contains an optimal schedule. Pruning rules discard only nodes that cannot lead to a better solution:

1) Lower bound pruning (Lemma 5.1): if $LB \geq UB$, no descendant can improve the best known solution.
2) Dominance pruning (Lemma 5.2): if a node is dominated, it cannot lead to a better solution than one already explored.
3) Resource infeasibility: if no feasible start time exists, the node has no feasible completions.

None of these discard the optimal schedule node.

Optimality: When a leaf node (complete schedule) is found, its makespan C is compared to UB. Initially UB is an upper bound (sum of all durations). Whenever a better schedule is found, UB decreases. Upon termination, UB equals the minimum makespan among all explored complete schedules. By completeness, this minimum equals OPT.

Theorem 5.4 (Comparison of Lower Bounds). Let $LB_{\text{workload}} = \max_k \lceil (\sum_{i \in U} r_{ik} d_i)/R_k \rceil + \text{current\_time}$. Then for any partial schedule S:

$$LB_{\text{cut}}(S) \geq LB_{\text{workload}}(S).$$

Moreover, there exist instances where the inequality is strict.

Proof. By Lemma 4.2, $\Phi_k \geq \lceil (\sum_i r_{ik} d_i)/R_k \rceil$. Adding current_time preserves the inequality. For strictness, see Mohring et al. (2003, Section 4.2) for an explicit counterexample.

6. Computational Experiments

6.1. Implementation and Hardware

The algorithm was implemented in Python 3.10 with Numba just-in-time compilation. Max-flow computations use Dinic's algorithm [15]. The Ford algorithm is called at each node. Experiments ran on an Intel Core i5-8250U processor with 8 GB RAM under Windows 10. We used the standard PSPLIB benchmark sets [32]: j30 (480 instances, 30 activities, 4 resources) and j60 (480 instances, 60 activities). A time limit of 600 seconds was imposed for j60.

6.2. Comparison with Classical Bounds

Table 1 compares our min-cut bound against the simple resource workload bound on a representative subset of j60 instances.

Table 1. *Comparison of Lower Bounds on j60 Instances (Average Values).*

Bound Type	Avg. LB (Root)	Max LB (Root)	Nodes Reduced (%)
Resource Workload	52.3	78	—
Min-Cut	61.7	89	47.2

6.3. Overall Results

Table 2 reports results on j30. Our algorithm solves all 480 instances optimally with average CPU time 2.0 seconds. For comparison, we also report results from Demeulemeester & Herroelen (1992) [7] and CPLEX 22.1.

Table 2. *Results on PSPLIB j30 (480 Instances).*

Method	Solved (%)	Avg. CPU (s)	Max Time (s)
Demeulemeester & Herroelen (1992)	100	1.9	10
CPLEX 22.1	100	0.9	5
Our Algorithm (Ford + Min-Cut)	100	2.0	9

Table 3 shows results on j60. Within 600 seconds, our algorithm solves 89% of instances optimally, compared to 78% with the simple resource workload bound.

Table 3. *Results on PSPLIB j60 (480 Instances, 600s Limit).*

Method	Solved (%)	Avg. Gap (%)	Avg. CPU (s)
Our Algorithm (Simple Bound)	78.1	4.2	142
Our Algorithm (Min-Cut Bound)	89.2	1.8	98
CPLEX 22.1	96.5	0.7	89

6.4. Discussion

The min-cut bound significantly improves performance: 11% more j60 instances solved optimally, with a 31% reduction in average CPU time. The gap between our algorithm and CPLEX (89% vs 96.5%) is now much smaller. The min-cut bound is computationally more expensive than the simple workload bound (max-flow vs. arithmetic), but the reduction in search tree size more than compensates.

7. Conclusion and Future Work

We have presented a new exact branch-and-bound algorithm for the Resource-Constrained Project Scheduling Problem that combines the Ford algorithm for precedence constraints with a minimum cut lower bound for resource constraints. The min-cut bound significantly improves bound quality, leading to a 31% reduction in solution time and an 11% increase in solved j60 instances compared to the simple workload bound. We provided complete rigorous proofs of correctness, including lemmas establishing admissibility of the bounds, dominance, termination, and optimality.

Future work includes:

1) Extension to generalized precedence relations [12]
2) Integration with a more sophisticated branching rule using machine learning [24]
3) Parallelization using work stealing [30]
4) Testing on the larger j120 benchmark set
5) Adaptation to stochastic [28] and robust [29] RCPSP variants

Our source code is available upon request from the corresponding author for reproducibility.

ORCID

0000-0002-6599-2157 (Aboubakary Diakhaby)

Abbreviations

CPM	Critical Path Method
CP	Critical Path
CPU	Central Processing Unit
DAG	Directed Acyclic Graph
EST	Earliest Start Time
GNN	Graph Neural Network
LST	Latest Start Time
MIP	Mixed-Integer Linear Programming
PSPLIB	Project Scheduling Problem Library
RCPSP	Resource-Constrained Project Scheduling Problem

Acknowledgments

The authors thank the authorities of the Republic of Guinea and in particular the Minister in charge of Higher Education for the creation of the West African Institute of Mathematics which serves as an environment for this work. This research received no specific grant from funding agencies.

Author Contributions

Hawa Kaba Bah: Data curation, Project administration, Software, Writing – original draft

Amadou Oury Balde: Formal Analysis, Investigation, Visualization

Ibrahima Bah: Conceptualization, Validation, Writing – review & editing

Aboubakary Diakhaby: Conceptualization, Methodology, Supervision, Writing – review & editing

Data Availability Statement

The PSPLIB benchmark instances are publicly available at https://www.om-db.wi.tum.de/psplib/ The source code of our algorithm is available upon request from the corresponding author.

Conflicts of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] L. R. Ford, Network flow theory, RAND Paper P-923, 1956.

[2] J. Blazewicz, J. K. Lenstra, A. H. G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, *Discrete Applied Mathematics* 5 (1983) 11-24. https://doi.org/10.1016/0166-218X(83)90012-4

[3] P. Brucker, A. Drexl, R. Mohring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, classification, methods, *European Journal of Operational Research* 112 (1999) 3-41. https://doi.org/10.1016/S0377-2217(98)00204-5

[4] E. Demeulemeester, W. Herroelen, *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers, 2002. https://doi.org/10.1007/b101924

[5] S. Hartmann, D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem, *European Journal of Operational Research* 207 (2010) 1-14. https://doi.org/10.1016/j.ejor.2009.11.005

[6] R. Kolisch, S. Hartmann, Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis, in: *Project Scheduling*, Springer, 1999, pp. 147-178. https://doi.org/10.1007/978-1-4615-5533-9_7

[7] E. Demeulemeester, W. Herroelen, A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, *Management Science* 38 (1992) 1803-1818. https://doi.org/10.1287/mnsc.38.12.1803

[8] E. Demeulemeester, W. Herroelen, A branch-and-bound procedure for the preemptive resource-constrained project scheduling problem, *European Journal of Operational Research* 90 (1997) 334-348. https://doi.org/10.1016/0377-2217(95)00342-8

[9] A. Mingozzi, V. Maniezzo, S. Ricciardelli, L. Bianco, An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation, *Management Science* 44 (1998) 714-729. https://doi.org/10.1287/mnsc.44.5.714

[10] A. Sprecher, Scheduling resource-constrained projects competitively at modest memory requirements, *Management Science* 46 (2000) 710-723. https://doi.org/10.1287/mnsc.46.5.710.12048

[11] A. Sprecher, A competitive branch-and-bound algorithm for the resource-constrained project scheduling problem, *European Journal of Operational Research* 90 (1996) 384-394.

[12] B. De Reyck, W. Herroelen, A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations, *European Journal of Operational Research* 111 (1998) 152-174. https://doi.org/10.1016/S0377-2217(97)00348-2

[13] A. Fest, R. H. Mohring, F. Stork, M. Uetz, Resource-constrained project scheduling with time windows: A branching scheme based on dynamic release dates, TU Berlin Report 596, 1998.

[14] R. H. Mohring, A. S. Schulz, F. Stork, M. Uetz, Solving project scheduling problems by minimum cut computations, *Management Science* 49 (2003) 330-350. https://doi.org/10.1287/mnsc.49.3.330.12737

[15] E. A. Dinic, Algorithm for solution of a problem of maximum flow in a network with power estimation, *Soviet Mathematics Doklady* 11 (1970) 1277-1280.

[16] A. V. Goldberg, R. E. Tarjan, A new approach to the maximum-flow problem, *Journal of the ACM* 35 (1988) 921-940. https://doi.org/10.1145/48014.61051

[17] O. Koné, C. Artigues, P. Lopez, M. Mongeau, Event-based MILP models for resource-constrained project scheduling problems, *Computers & Operations Research* 38 (2011) 3-13. https://doi.org/10.1016/j.cor.2009.12.011

[18] P. Laborie, Algorithms for propagating resource constraints in AI planning and scheduling, *Artificial Intelligence* 143 (2003) 151-188. https://doi.org/10.1016/S0004-3702(02)00362-4

[19] P. Laborie, J. Rogerie, P. Shaw, P. Vilim, IBM ILOG CP Optimizer for scheduling, *Constraints* 23 (2018) 210-250. https://doi.org/10.1007/s10601-018-9281-x

[20] Hexaly, Benchmark: Hexaly vs OR-Tools on the Resource-Constrained Project Scheduling Problem, https://www.hexaly.com/benchmarks, accessed 2024

[21] X. Li, J. Zhang, S. Wang, Deep reinforcement learning for resource-constrained project scheduling, *IEEE Transactions on Automation Science and Engineering* 17 (2020) 1470-1482. https://doi.org/10.1109/TASE.2020.2974920

[22] Z. Wang, X. Chen, L. Zhang, Graph neural networks for resource-constrained project scheduling, *Computers & Operations Research* 136 (2021) 105481. https://doi.org/10.1016/j.cor.2021.105481

[23] X. Chen, Y. Liu, Y. Zhang, Reinforcement learning for resource-constrained project scheduling: A review and new directions, *Computers & Industrial Engineering* 170 (2022) 108271. https://doi.org/10.1016/j.cie.2022.108271

[24] Y. Zhang, L. Liu, H. Zhou, Learning to branch for resource-constrained project scheduling, *INFORMS Journal on Computing* 35 (2023) 612-628. https://doi.org/10.1287/ijoc.2023.1291

[25] M. Davari, E. Demeulemeester, A novel branch-and-bound algorithm for the chance-constrained RCPSP, *International Journal of Production Research* 57 (2019) 1265-1282. https://doi.org/10.1080/00207543.2018.1504245

[26] Y. Alipouri, A resource flow-based branch-and-bound algorithm to solve fuzzy stochastic resource-constrained project scheduling problem, *Soft Computing* 25 (2021) 1-17. https://doi.org/10.1007/s00500-020-05527-3

[27] S. Creemers, Minimizing the expected makespan of a project with stochastic activity durations under resource constraints, *Journal of Scheduling* 18 (2015) 263-273. https://doi.org/10.1007/s10951-014-0391-z

[28] S. Rostami, S. Creemers, R. Leus, New strategies for stochastic resource-constrained project scheduling, *Journal of Scheduling* 21 (2018) 349-365. https://doi.org/10.1007/s10951-016-0505-x

[29] M. E. Bruni, P. Di Puglia Pugliese, P. Beraldi, F. Guerriero, An adjustable robust optimization model for the resource-constrained project scheduling problem with uncertain activity durations, *Omega* 71 (2017) 66-84. https://doi.org/10.1016/j.omega.2016.09.009

[30] R. D. Blumofe, C. E. Leiserson, Scheduling multithreaded computations by work stealing, *Journal of the ACM* 46 (1999) 720-748. https://doi.org/10.1145/324133.324136

[31] W. Herroelen, B. De Reyck, E. Demeulemeester, Resource-constrained project scheduling: A survey of recent developments, *Computers & Operations Research* 25 (1998) 279-302. https://doi.org/10.1016/S0305-0548(97)00056-X

[32] PSPLIB - Project Scheduling Problem Library, https://www.om-db.wi.tum.de/psplib/