Branch and Price for Job-Shop Scheduling with Time-Dependent Costs and Cardinality Resource Constraints

Marouane Felloussi[1,2], Mohammed Ghannam[2], João Dionísio[2,3], Paolo Gianessi[1], and Xavier Delorme[1]

[1] Mines Saint-Etienne, Univ. Clermont Auvergne, INP Clermont Auvergne, CNRS, UMR 6158 LIMOS, F-42023 Saint-Etienne, France
[2] Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
[3] Faculdade de Ciências, Universidade do Porto, rua do Campo Alegre, Porto, Portugal

January 14, 2026

Abstract. Motivated by settings with fluctuating energy prices and peak-capacity limits, we study a discrete-time job-shop scheduling problem with time-varying operating costs and constraints on the number of simultaneously active machines. The objective is to minimize total operating cost. We develop a branch-and-price algorithm based on an extended formulation over job schedules, where the pricing subproblem is solved exactly by a forward dynamic program. We introduce a hierarchical branching scheme that enforces decisions directly in pricing, combining a problem-specific variable selection rule, strong branching, and pseudo-costs; propagation then exploits the problem structure to derive additional fixings. On the primal side, we develop an exact tree search over the restricted column pool with a structure-aware lower bound, as well as a large neighborhood search in which neighborhoods are explored efficiently via the pricing dynamic program. Computational experiments on 504 instances show that our approach solves 77% of instances to optimality within the time limit, substantially outperforming an aggregated and a disaggregated compact time-indexed formulations.

Keywords: Job-shop scheduling · Time-dependent operating costs · Resource constraints · Branch and price · Integer programming

1 Introduction

Time-dependent operating costs and resource limits naturally arise in shop-floor scheduling, driven, for instance, by fluctuating operating rates such as energy prices and contractual peak-capacity constraints. In such settings, schedules are evaluated not only by traditional metrics like completion time, but also by how resource usage is allocated over time.

The Job-Shop Scheduling Problem (JSSP) is a well-known NP-hard problem [12]. It consists of sequencing operations on dedicated machines, where each

operation must be processed in a predefined order without overlapping. Operations are processed non-preemptively, must respect job precedences, and cannot overlap on the same machine. We study a discrete-time JSSP variant with time-dependent operating costs and per-time-step cardinality limits that bound the number of machines that may be active simultaneously.

In such settings, time-indexed formulations not only represent a natural modeling choice, but also typically yield strong linear relaxations [2]. Their main drawback, however, is size: they scale poorly with the horizon length, leading to pseudo-polynomial models that are often difficult to solve directly. Period-indexed variants [10] apply when costs and/or resource limit profiles are piecewise-constant (e.g., Time-of-Use pricing); hence, they can reduce model size as the number of periods is orders of magnitude smaller than the number of time steps. However, they do not address settings where costs and limits vary at unit-time resolution. Decomposition-based extended formulations have proven effective in parallel-machine environments [21], but their applications to shop-scheduling remain limited and computationally challenging [15]. Moreover, this problem class can exhibit several practical difficulties documented for branch and price [14], notably master problem degeneracy and dual oscillations.

The solution approach we propose is an exact branch-and-price framework. It is based on an extended formulation over job schedules, previously used in similar precedence-constrained settings [9, 15]. The formulation (Sec. 2) can be viewed as a Dantzig–Wolfe reformulation of a disaggregated time-indexed model [2], where the pricing integer subproblem (ISP) satisfies the integrality property [14]. The reformulation, therefore, preserves the dual bounds of the disaggregated model. In our setting, however, the pricing has a network-flow structure and can be solved exactly by a fast forward dynamic program (Sec. 3). This delegates horizon-dependent combinatorial work to the oracle and makes repeated calls inexpensive, enabling components that rely on frequent pricing evaluations.

On top of this foundation, we propose (Sec. 4) an effective problem-specific variable selection rule for branching and embed it in a hierarchical scheme that adapts standard elements from the MIP literature. We enforce the resulting branching decisions within the ISP and combine problem-structure information with a hybrid of strong branching and pseudo-cost branching. Local propagation exploits disjunctive, precedence, and time-capacity constraints to derive additional fixings. We design primal heuristics (Sec. 5) around these structural implications and the fast pricing oracle, enabling efficient construction of feasible solutions. Computational experiments (Sec. 6) on a large and varied benchmark set compare the resulting approach against an aggregated and a disaggregated compact time-indexed formulation, and an ablation study analyzes the contribution of individual components from both primal and dual perspectives.

2 Problem and Extended Formulation

We consider a set of machines $\mathcal{M}$, and a set of jobs $\mathcal{J}$ over a discrete time horizon $\mathcal{T}$. Each job $j \in \mathcal{J}$ consists of a sequence of $|\mathcal{M}|$ operations and visits

each machine exactly once, with permutations of $\mathcal{M}$ defining different processing orders. The operation of j on m is denoted by (j, m). We write $(j, m) \prec (j, m')$ if m precedes m' in the machine permutation of j, therefore (j, m) must complete before (j, m') can start. The processing time of (j, m) is a constant value $q_{j,m}$. At each time step $t \in \mathcal{T}$, a resource limit in the form of a cardinality constraint specifies that at most $\overline{M}_t$ machines may be active at once. We also consider that executing (j, m) at time t incurs a cost $g^t_{j,m}$.

Time-dependent costs and cardinality constraints naturally lead to time-indexed formulations, in which binary variables (e.g.) indicate whether an operation is processed at a time step. These variables can be used to express the job-shop requirements and the cardinality limit on concurrent machine activity; non-singular transformations of the same variable space yield equivalent time-indexed models, including aggregated and disaggregated precedence representations, the latter yielding stronger LP relaxations [2]. While these formulations offer strong dual bounds, their pseudo-polynomial size relative to the time horizon often leads to large ILPs that scale poorly as the horizon increases.

To mitigate this explicit size, the problem is decomposed into job schedules. Each schedule assigns a job's operations to discrete time steps while respecting precedence and non-preemption. Let $\mathcal{P}_j$ denote the set of feasible schedules for j, and let $\mathbf{P} \coloneqq \bigcup_{j \in \mathcal{J}} \mathcal{P}_j$. Each column $p \in \mathcal{P}_j$ is represented by a binary profile $a^p_{m,t} \in \{0, 1\}$ indicating whether the schedule p uses machine m at time t, and its induced cost under the time-dependent rates $c_p \coloneqq \sum_{m \in \mathcal{M}} \sum_{t \in \mathcal{T}} g^t_{j,m} a^p_{m,t}$. This yields the following set-partitioning formulation:

$$\min \quad \sum_{p \in \mathbf{P}} c_p \lambda_p, \tag{1.1}$$

$$\text{s.t.} \quad \sum_{p \in P^j} \lambda_p = 1, \qquad \forall j \in \mathcal{J}, \tag{1.2}$$

$$\sum_{p \in \mathbf{P}} a^p_{m,t} \lambda_p \leqslant 1, \qquad \forall m \in \mathcal{M}, \forall t \in \mathcal{T}, \tag{1.3}$$

$$\sum_{p \in \mathbf{P}} \sum_{m \in \mathcal{M}} a^p_{m,t} \lambda_p \leqslant \overline{M}_t, \qquad \forall t \in \mathcal{T}, \tag{1.4}$$

$$\lambda^p \in \{0, 1\}, \qquad \forall p \in \mathbf{P}, \tag{1.5}$$

where (1.2)–(1.4) are the job assignment, machine disjunction, and resource capacity constraints. Formulation (1) is indexed by the complete sets $\mathcal{P}_j$ of feasible job schedules, which, due to their large size, typically make it impractical to manage the formulation explicitly. Instead, the LP relaxation is solved via column generation (CG) at the nodes of a branch-and-bound tree, yielding a branch-and-price algorithm [8]. In CG, the LP relaxation is obtained by relaxing integrality on λ^p, defining a master problem (**MP**) over the column set $\mathbf{P}$. Given its high cardinality, the algorithm works on a restricted subset $\mathbf{P'} \subset \mathbf{P}$, defining the restricted master problem (**RMP**). CG is a primal-simplex-type algorithm and begins with an initialization phase that establishes primal feasibility. Thereafter, each iteration yields a primal-dual solution $(\hat{\lambda}, [\hat{\alpha}, \hat{\beta}, \hat{\gamma}])$ of the (**RMP**), and an

integer pricing subproblem is solved to search for a column with negative reduced cost with respect to the dual values. If such a column is found, it is added to $\mathbf{P}'$ and the (**RMP**) is re-optimized; otherwise, dual feasibility is attained and the solution is optimal for the (**MP**).

3 Pricing by Dynamic Programming

Let $\hat{\alpha}$, $\hat{\beta}$, and $\hat{\gamma}$ denote the optimal dual values for constraints (1.2), (1.3), and (1.4), respectively, obtained after solving the (**RMP**) at a CG iteration. The set-partitioning constraints (1.2) imply that $\lambda^p \leqslant 1, \forall p \in \mathbf{P}$, so the variable bounds (1.5) can be relaxed to $\lambda^p \geqslant 0$. In the following, we reason about a fixed job j. Let $\hat{c}^t_{j,m} \coloneqq g^t_{j,m} - \hat{\beta}_{m,t} - \hat{\gamma}_t$. The reduced cost of a column $p \in \mathcal{P}_j$ equals $-\hat{\alpha}_j + \sum_{m,t} \hat{c}^t_{j,m} a^p_{m,t}$, and pricing for j reduces to solving $\min_{p\in\mathcal{P}_j} \sum_{m,t} \hat{c}^t_{j,m} a^p_{m,t}$. A feasible schedule assigns a completion time t to all operations. Let $i(j,m)$ denote the position of (j,m) in the processing sequence of job j. This assignment must satisfy non-preemptive processing–where operation (j,m) executes over $q_{j,m}$ consecutive time steps–and respect precedence constraints by ensuring that (j,m) may start only after the completion of the operation in position $i(j,m)-1$. For notational convenience, we index the operations of j by their position i; accordingly, we write $q_i \equiv q_{j,m}$ and $\hat{c}^\tau_{j,m} \equiv \hat{c}^\tau_i$. Finding a minimum-cost partial schedule can be described as a dynamic program (DP) on the state (i,t), where i is the operation index and t its completion time. Let $d^\star_j(i,t)$ denote the minimum-cost of any partial schedule that completes operation i of job j exactly at time t, and let $S^t_i \coloneqq \sum_{\tau=t-q_i+1}^{t} \hat{c}^\tau_i$. At a state (i,t), two transitions are possible:

- *Idle transition:* no operation is processed at time t; $d^\star_j(i,t) \leftarrow d^\star_j(i,t-1)$.
- *Processing transition:* operation i completes at time t after executing over time steps $\{t-q_i+1,\ldots,t\}$; $d^\star_j(i,t) \leftarrow d^\star_j(i-1,t-q_i) + S^t_i$.

With the convention $d^\star_j(\cdot,\cdot) = +\infty$ for infeasible indices and $d^\star_j(0,t) = 0$ for all admissible t, the recurrence is

$$d^\star_j(i,t) = \min\Big\{ d^\star_j(i,t-1), d^\star_j(i-1,t-q_i) + S^t_i \Big\},$$

which is non-increasing in t. Therefore, the minimum reduced cost of any complete pattern of j equals $d^\star_j(|\mathcal{M}|,|\mathcal{T}|)$. Equivalently, this is a shortest-path problem on a directed acyclic graph whose nodes are pairs (i,t), and whose arcs represent idle transitions of zero length and processing transitions of length S^t_i. Since it is the convex hull of incidence vectors of these paths, the induced path polytope is integral (e.g., [14]). Therefore, this reformulation does not strengthen the LP bound relative to a compact formulation with disaggregated precedence constraints [2]. The advantage of the reformulation lies instead in reducing the size of the problem and shifting computational effort to fast pricing oracles.

Two standard refinements further accelerate the dynamic program. First, the earliest start times and latest completion times are computed for each operation based on the precedence graph and processing durations. The resulting time

windows restrict the admissible values of t in the DP state space. Second, for each pair (i,t), the window sums S_i^t satisfy the recurrence $S_i^t = S_i^{t-1} - \hat{c}_i^{t-q_i} + \hat{c}_i^t$, so that all required values can be updated in constant time per increment of t.

Dual stabilization and complementary pricing. For any dual optimal solution of (**MP**), $\alpha_j \geqslant \min_{p \in \mathcal{P}_j} c_p$ holds given the non-positivity of $\beta_{m,t}$ and γ_t. This constraint is enforced in the dual space by adding the corresponding variable in the primal. Furthermore, since all pricing subproblems are solved exactly, the parameter-free dual stabilization scheme of [20] is used to stabilize the dual sequence and improve the column generation convergence. In addition, at each column generation iteration at the root node, we perform an additional randomized pricing call. Given the current (**RMP**) solution $\hat{\lambda}$, we forbid execution at (j,m,t) with probability $\sum_{p \in \mathcal{P}_j} a_{m,t}^p \hat{\lambda}_p$, which acts complementary to the active columns.

4 Branching and propagation

We solve the integer problem using a branch-and-price algorithm [8], where column generation solves the LP relaxation at each node of the tree while respecting branching constraints. Solving (**MP**) at a node yields an LP-optimal solution. If $\lambda^\star$ is integral, we update the incumbent if improving and prune the node. If $\lambda^\star$ is fractional, we prune the node if the dual bound is greater than or equal to the primal bound; otherwise, we branch in either the master or integer subproblem (ISP) space, and update the incumbent if the solution induced in the original space is integral. When branching constraints are added directly to the pricing problem, the LP bound of a node is computed over the convex hull of $\mathbf{F} \cap \mathcal{B}_n$, where $\mathbf{F}$ denotes the subproblem feasible set and $\mathcal{B}_n$ is the set of branching constraints along the path to n. In contrast, branching on the master solves the LP over $\text{conv}(\mathbf{F}) \cap \mathcal{B}_n$, typically resulting in weaker bounds. In preliminary experiments, branching on the master (e.g., resource usage) led to large trees. Among ISP branching rules, branching on starting times was ineffective. Branching on the original binary variables $z_{j,m}^t = \sum_{p \in P^j} a_{m,t}^p \lambda_p$, denoting whether operation (j,m) is processing at time step t, consistently led to faster proofs of optimality. The two branches $z_{j,m}^t = 1$ and $z_{j,m}^t = 0$ correspond, respectively, to forcing or forbidding the processing transition of operation (j,m) at time t. Since forcing $z_{j,m}^t = 1$ is equivalent to forbidding the processing transition at all $t' \neq t$, and $z_{j,m}^t = 0$ directly forbids it at t, both branches reduce to forbidding a set of processing transitions, which are excluded when computing the DP recurrence. We note that integrality of the z variables is equivalent to that of the λ variables: for each job j, the set partitioning constraint implies that the vector $(z_{j,m}^t)_{m,t \in \mathcal{M} \times \mathcal{T}}$ is a convex combination of distinct 0-1 profiles, which is binary only if all weight is placed on a single profile. Hence, in our setting, it is sufficient to check the integrality of λ during branching. Finally, branching constraints may lead to an infeasible RMP at certain nodes. In that case, the same (**RMP**) initialization approach is invoked to restore feasibility or prune the node by proving the infeasibility of the current subproblem.

Most-dispersed branching. In integer feasible solutions, operations execute over a contiguous time block. The branching rule exploits this by choosing an operation whose execution is most dispersed in a fractional solution $(\lambda^\star, z^\star)$ and its most-used time step. For each (j,m), we define $F_{j,m} \coloneqq \{t \in \mathcal{T} : 0 < z^{\star t}_{j,m} < 1\}$ and choose $(j,m) = \arg\max_{(j,m)\in\mathcal{M}\times\mathcal{T}} |F_{j,m}|$. Then, we take $t = \arg\max_{t\in F_{j,m}} z^{\star t}_{j,m}$.

Reliable pseudo-cost branching. In MIP, a standard variable selection strategy is pseudo-cost branching [3]. For each candidate branching variable, pseudo-costs estimate dual-bound improvement per unit of bound change. This yields a problem-agnostic and instance-adaptive ranking of branching candidates. Since pseudo-costs are initially unavailable, the most-dispersed rule is first used to filter branching candidates. On this reduced set, strong branching without pricing is performed, and the variable with the best combined dual bound improvement is selected. The resulting pseudo-costs are included in the history but with a reduced weight, since they are based on a single RMP solve. We apply this hierarchical branching scheme only near the root node. Furthermore, a reliability threshold specifies how many observations are required before a variable's pseudo-cost is usable; if, at a given node, no fractional variable has reliable pseudo-costs, additional strong-branching iterations are executed within a prescribed budget. At any node where at least one fractional variable has reliable pseudo-costs, the branching variable is chosen by the pseudo-cost rule; otherwise, the algorithm falls back to the initial most-dispersed rule. These pseudo-cost scores are complemented with aggregated pseudo-cost information computed over variable sets: average dual gains are maintained by operation (j,m) and per time t, and combined with the variable-level pseudo-cost through a weighting parameter. The rationale is that branching on $z^t_{j,m}$ also restricts variables related to operation (j,m) and time step t separately.

Propagation and inference scores. To ensure consistency with the branching decisions, we fix to zero any column that violates $\mathcal{B}_n$. Furthermore, we derive the following fixings from the problem structure:

Job and machine disjunction: Each job occupies one machine at a time, and each machine processes at most one job at a time. If $\{z^t_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j,m'} = 0\}$ for all $m' \neq m$; $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j',m} = 0\}$ for all $j' \neq j$.

Precedence: Preceding operations cannot be active after a fixed processing time step. If $\{z^t_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^{t'}_{j,m'} = 0\}$ for $(j,m') \prec (j,m)$ and $t' > t$. A similar argument holds for succeeding operations.

Non-preemption: All time steps between two active ones must also be active. If $\{z^{t_1}_{j,m} = 1\}, \{z^{t_2}_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j,m} = 1\}$ for $t_1 < t < t_2$.

Non-preemption: All time steps outside a contiguous active block must be inactive. If $\{z^{t_1}_{j,m} = 0\}, \{z^{t_2}_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j,m} = 0\}$ for $t < t_1$ if $t_1 < t_2$, and for $t > t_2$ if $t_2 < t_1$.

Active machines limit: If the active machine limit is reached at a time step, all remaining machines must be idle. For $t \in \mathcal{T}$ and $\mathcal{M}' \subset \mathcal{M}$ with $|\mathcal{M}'| = \overline{M}_t$, if $\forall m \in \mathcal{M}', \exists j \in \mathcal{J}, \{z^t_{j,m} = 1\} \in \mathcal{B}_n$, then $\mathcal{B}_n \leftarrow \mathcal{B}_n \cup \{z^t_{j,m'} = 0\}$ for all $m' \in \mathcal{M} \setminus \mathcal{M}'$ and $j \in \mathcal{J}$.

Upon each branching decision, $\mathcal{B}_n$ is updated immediately by applying the rules above. This propagation routine is also invoked during strong branching tests. Furthermore, as the formulation contains many clique inequalities and set-partitioning equalities over large sets, a large number of logical deductions can be obtained from branching decisions. This count describes an inference score, as an additional measure of the impact of that branching decision. This score is then combined with pseudo-costs as in hybrid pseudo-cost inference branching [1].

Lagrangian bound and early branching. With exact pricing, we compute a Lagrangian lower bound LB_{lag} on the value of (**MP**) [16]. Furthermore, a preprocessing step scales the data to obtain an integer objective. This allows using $\lceil LB_{\text{lag}} \rceil$ as a valid lower bound. Early branching [19] at a node (i.e., before column generation convergence) can be performed without losing relaxation strength if the LP objective of (**RMP**) verifies $\lceil z_{\text{RMP}} \rceil = \lceil LB_{\text{lag}} \rceil$.

5 Primal heuristics

At any node of the branch-and-price (B&P) tree, primal solutions are obtained by combining columns from the current pool into a feasible solution. Generic MIP heuristics [6] apply, but they are agnostic to the column generation process. In fact, they do not distinguish whether the absence of an incumbent is due to i) intrinsic heuristic limitations, or ii) the absence of feasible incumbents in the current column set. A straightforward approach is the restricted master heuristic (or price and branch), in which the (**RMP**) at a node is solved as a 0-1 MIP after enforcing integrality on the master variables, which can answer both of the aforementioned questions. However, the resulting problem is a large-scale set-partitioning integer program with additional clique- and knapsack-type constraints. Preliminary experiments with state-of-the-art MIP and CP-SAT solvers show that this problem is computationally challenging, even in its decision form. To solve this problem, we propose an exact branch-and-bound scheme with problem-specific bounding and a large-neighborhood search, both tailored to exploit the problem structure.

Exact tree search and bounding. We solve the 0-1 (**RMP**) over the column pool available at the branch-and-price node where the heuristic is invoked, via a depth-first search (DFS) that assigns one column per job in a fixed order. At depth k, a column is selected for the k-th job, the set of occupied (m, t) pairs and the time-wise count of active machines are updated, and the search recurses. We prune nodes using a cutoff bound and an additive lower bound obtained by completing the remaining jobs with their cheapest compatible columns.

We first order jobs in decreasing number of available columns to propagate tighter set-partitioning constraints earlier in the tree. Then, for each job j, we sort the column set $\mathcal{P}_j$ by increasing cost. A DFS node is represented by $(k, \mathcal{Z}, X, u, v)$ where k is the current job index in the fixed order, $\mathcal{Z}$ the set of fixed columns in the ancestors, $X \subset \mathcal{M} \times \mathcal{T}$ the set of occupied (m, t) pairs, $u = (u_t)_{t \in \mathcal{T}}$ encodes the number of active machines over $\mathcal{T}$, and v is the partial cost of the columns in $\mathcal{Z}$. Given a node $(k, \mathcal{Z}, X, u, v)$ and a cutoff bound z^{cut}:

1. If $v \geqslant z^{\text{cut}}$, prune the node.
2. If $k = |\mathcal{J}|$ then update incumbent with v and $\mathcal{Z}$ if improving current best.
3. Otherwise, for each column $p \in \mathcal{P}_k$ (in increasing cost order):
   (a) Lower bound: if $\texttt{LB} \geqslant z^{\text{cut}}$ then prune,
   $\texttt{LB}(k, X, u) = v + c_p + \sum_{j>k} \min_{q \in \mathcal{P}_j} \{c_q : q \text{ compatible with } p \text{ and } (X, u)\}$.
   (b) Check feasibility: $\forall (m,t)$ s.t. $a^p_{m,t} = 1 : (m,t) \notin X, u_t + \sum_m a^p_{m,t} \leqslant \overline{M}_t$ otherwise continue with the next column.
   (c) Update state: $X \leftarrow X \cup \{(m,t) : a^p_{m,t} = 1\}$, $u_t \leftarrow u_t + \sum_m a^p_{m,t}$.
   (d) Process the next job (i.e., child node in DFS).
   (e) Restore state on backtracking.

The search is initialized at the root with $\mathcal{Z} = \varnothing, X = \varnothing, u = \mathbf{0}, v = 0$. We start from the first job in the sorted order and set z^{cut} to the current incumbent value (or $+\infty$ if none exists). The search either runs to optimality over the current column pool or stops upon finding a feasible improving solution. We optionally prefilter the column sets by restricting to those active in the last LP solution; thus, the procedure reduces to an optimal rounding heuristic (RENS-mode) [5]. Near the leaves, we cache partial solutions and reuse them as initial seeds for the large neighborhood search (LNS) heuristic described below.

Large Neighborhood Search: LNS operates on available integer solutions found throughout the B&P or by extending partial solutions during the search tree. For a complete incumbent λ^{inc} and a destroy set $D \subset \mathcal{J}$, we define the neighborhood

$$\mathcal{N}(D, \lambda^{\text{inc}}) \coloneqq \left\{ \lambda \in \{0,1\}^{|\mathcal{P}|} \;:\; \begin{array}{l} \displaystyle\sum_{p \in \mathcal{P}_j} \lambda_p = 1, \forall j \in \mathcal{J}, \\ \lambda_p = \lambda^{\text{inc}}_p, \quad \forall j \in \mathcal{J} \setminus D,\ \forall p \in \mathcal{P}_j. \end{array} \right\},$$

subject to the same clique and cardinality constraints. A similar construction is used when starting from a partial solution: a subset of jobs is kept fixed to their current columns, and the remaining jobs are reoptimized over the corresponding neighborhood. Destroy-repair cycles are performed until no improvement is found or a limit is reached, and any improving full solution is accepted as a new incumbent. The destroy and repair step exploits the speed of the integer subproblem to generate improved columns for the unfixed jobs; algorithmically, both pricing and LNS call the same oracle, differing only in the objective and in the additional fixings induced by neighborhood definition. The exact tree search and the LNS heuristics are disabled after no primal improvement has been observed for a prescribed number of B&P nodes (primal tailing-off). In the exact search, the number of explored nodes is bounded, whereas the LNS is stopped after a fixed number of consecutive iterations without improvement.

6 Computational experiments

Instances. We evaluate our algorithm on 504 instances generated following the scheme in [18]. The set is based on seven job-shop instances with $|\mathcal{J}| \in \{5, 6, 7, 8\}$ and $|\mathcal{M}| \in \{6, 7, 8, 10\}$, and varying operation processing time ranges. They are

extended with operational costs of the form $g^t_{j,m} = \varphi_m \kappa^t$, where φ_m models machine-dependent intensity and κ^t is a time-dependent cost profile, consistent with industrial settings where energy or resource costs vary by machine type and time step. We consider three value sets for φ inspired by knapsack (KP) sets [13]: small and large coefficient KP, three-coefficient KP, and arithmetic-sequence KP. For κ, we consider two time-of-use profiles inspired by electricity tariff structures [10, 22], and one uniformly random profile. The maximum number of active machines is fixed for all $t \in \mathcal{T}$ to values in $\{|\mathcal{M}| - 1, |\mathcal{M}| - 2\}$, reflecting capacity constraints on simultaneous resource usage, common in power-limited manufacturing environments. Planning horizons are set with $|\mathcal{T}| = \lceil \omega C^\star \rceil$, with expansion factor $\omega \in \{1.2, 1.6, 2.0, 3.0\}$, and $C^\star$ is the makespan of the base instance under the cardinality constraint. $C^\star$ values range from 56 for the smallest instances to around 850 for the largest ones.

Setup and evaluation metrics. All experiments run single-threaded in exclusive mode on an Intel Xeon Gold 5122 CPU with 96 GB of RAM under a one-hour time limit. The implementation uses PySCIPOpt [17] with SCIP v9.2.4 [7] as the branch-and-price framework and SoPlex v7.1.6 as the LP solver. We use SCIP for its plugin-based architecture, which allows fine-grained control over the branch-and-price process. The pricing oracle is implemented in C, and all remaining algorithmic components are in Python. MIP- and expensive LP-based default SCIP primal heuristics are disabled in the branch-and-price approach, as their effectiveness relies on the generation of integer-compatible columns. We use Farkas pricing [11] to initialize the (**RMP**) or restore feasibility (if needed) after branching, dual simplex for the reoptimizations, and restrict the use of probability-based pricing to root-node column generation. The pseudo-costs obtained from strong branching are added to the history with a scaling factor of 0.8, whereas the variable-level pseudo-cost weighting parameter is set to 0.05. The DFS tree-search is executed at the root and subsequently until an incumbent is found; it is otherwise called every 50 nodes, with an additional RENS-mode call every 11 nodes. The DFS tree-search is capped at 10^6 nodes. LNS extensions are applied to the partial solutions found during the tree search with at least $|\mathcal{J}| - 2$ jobs, and are disabled after 10 consecutive calls without improvement. These parameters were set based on preliminary tuning.

In all result tables, $\#\mathbf{x}^\star$ denotes the number of instances solved to optimality and $\#\bar{\mathbf{x}}$ the number of instances for which a feasible solution was obtained. Performance is reported in terms of CPU time `T` (s) and the number of explored nodes #n. To mitigate the impact of outliers, aggregate results are computed using the shifted geometric mean (shift = 1). Unless stated otherwise, these aggregates are evaluated over the instances solved to optimality within the corresponding comparison set.

Branch-and-price performance. We compare the proposed branch-and-price approach (B&P), based on the extended (**E**) formulation, to aggregated (**A**) and disaggregated (**D**) time-indexed formulations [2], solved directly as integer programs. In Table 1, instances are categorized into subsets based on the time horizon length, determined by the value of $C^\star$.

Table 1: Comparison of the proposed B&P against the aggregated and disaggregated time-indexed B&B approaches.

Instances		Branch and Bound (**A**)				Branch and Bound (**D**)				Branch and Price (**E**)			
subset	#inst.	#$\mathbf{x}^\star$	#$\bar{\mathbf{x}}$	T	#n	#$\mathbf{x}^\star$	#$\bar{\mathbf{x}}$	T	#n	#$\mathbf{x}^\star$	#$\bar{\mathbf{x}}$	T	#n
small	144	131	144	94.4	98.5	131	144	134.5	7.0	130	144	15.5	14.3
medium	144	24	99	2226.4	691.7	75	114	1327.3	3.1	110	133	41.2	13.6
large	216	0	54	Timlim	-	41	78	1813.1	3.0	150	187	79.4	15.3
all	504	155	297	154.4	133.4	247	336	415.9	4.8	**390**	**464**	**38.5**	14.5

The results indicate that the proposed B&P is more reliable than the integer programs based on **A** and **D**, with a performance gap that widens with the time horizon. While the three approaches show comparable feasibility and optimality on small instances, B&P reaches comparable results faster while exploring a small tree. On medium and large instances, the compact formulations' performance deteriorates, whereas B&P achieves optimality more consistently. This performance gap reflects two complementary effects. First, both compact formulations exhibit slow LP relaxations that inflate per-node costs throughout the solve, an effect that compounds with the horizon. Second, while the LP relaxation of **E** matches that of **D** due to the integrality property, **E** solves each node relaxation faster thanks to the pricing oracle and dual stabilization; moreover, the proposed algorithm exploits the problem's structure through mechanisms such as the primal heuristic and propagation to keep the tree small. As for the compact formulations, **D** finds more feasible and optimal solutions than **A** but proves unreliable at scale: among the 168 instances where it fails to find a feasible solution, 54 terminate due to memory limits, whereas 66 fail to solve the root LP within the time limit. Root LP statistics further support these observations. Over all 504 instances, **E** solves 499 root relaxations within the time limit, against 338 for **D** and 391 for **A**. The average (shift=0.01) gap between the root LP bound and the optimal value, computed over instances where both values are available, is 0.01% for both formulations **E** and **D**, and 0.03% for formulation **A**.

Branch-and-price feature analysis. To test the performance of the proposed B&P features, we carry out an ablation study in a tighter-horizon regime where $\omega \in \{1.2, 1.6\}$. We omit the ablation results for $\omega \geqslant 2$, as the problem becomes a loose packing problem over job schedules, with variants solving nearly all instances in similar runtimes. We compare a baseline configuration (`basic`), which includes dual stabilization, probability-based complementary pricing, and the most-dispersed branching rule; a full configuration (`all`); and ablations of the latter (`all-X`), in which component `X` is disabled. The baseline components were essential to performance in our setting, so we include them by default and omit their ablation for brevity. Component `Pr` refers to propagation, `Ps` to pseudo-costs and strong branching, and `H` to the primal heuristics. For fair, paired comparisons, we report results on the intersection subsets all-feasible (all

Table 2: Comparison of the primal and dual features of the algorithm.

	all inst. (252)		all-feas. (166)		all-opt. (114)		
variant	$\#\bar{\mathbf{x}}$	$\#\mathbf{x}^\star$	PI	$\%\mathtt{T}^{\mathtt{1st}}$	$\%\mathtt{T}^{\mathtt{opt}}$	T	#n
basic	177	131	24	33	67	117.6	103
all-H	172	137	36	40	84	**72.3**	43
all	**212**	**138**	**17**	**21**	**86**	92.5	**36**

	all inst. (252)		all-opt. (115)	
variant	$\#\mathbf{x}^\star$	DI	T	#n
basic	131	3.3	113.8	102
all-Ps	127	**2.3**	147.2	87
all-Pr	**145**	2.6	102.6	52
all	138	2.8	**95.9**	**36**

variants find a feasible solution) and all-optimal (all variants prove optimality), noting that the latter can over-represent easier instances. We also report primal (PI) and dual (DI) integrals [4] relative to the best known bounds to summarize incumbent and bound trajectories over time. We use the relative times to first and optimal incumbents ($\%\mathtt{T}^{\mathtt{1st}}$ and $\%\mathtt{T}^{\mathtt{opt}}$) to separate early primal progress from the optimality proof effort. We analyze the ablations from complementary primal and dual perspectives; Table 2 reports the corresponding results.

Primal side: On the tighter-horizon instances, the full (`all`) configuration mainly improves primal effectiveness. It finds feasible solutions more often and earlier, maintaining higher-quality incumbents throughout the run (lower `PI`). On instances solved to optimality (all-opt.), the $\%\mathtt{T}^{\mathtt{opt}}$ values indicate that the optimal incumbent is typically reached late; the remaining proof effort then appears to come primarily from bound-based pruning of open nodes. We see a clear trade-off when isolating the heuristic (`all-H`): while removing it speeds up the search on solved instances, it reduces performance on the full set in terms of feasibility and tree size.

Dual side: Our study shows that the dual-side features improve the bound trajectory on average, as reflected by lower DI for the enhanced variants, but they differ sharply in how they trade pruning against overhead. Removing pseudo-costs and strong branching (`all-Ps`) is the only change that reduces effectiveness: it proves optimality least often and induces the largest increase in node counts, indicating that strong branching and the associated pseudo-cost collection are highly beneficial, with pruning gains that outweigh their overhead in the full-feature configuration. Importantly, `all-Ps` is also slower than `basic` because it retains the other expensive components (notably the heuristic), so it combines weaker branching guidance with much of the overhead.

While disabling propagation (`all-Pr`) increases the number of nodes and solving time on the all-optimal subset, it is the only ablation that proves optimality more than `all` over the full set. The limiting factor appears to be compounded per-node overhead rather than pruning power. With propagation enabled, our profiling indicates that the average pricing-oracle call is 48% slower; since pricing is invoked repeatedly throughout the solve, this slowdown accumulates quickly. It also impacts total strong-branching time, causing a 70% slowdown. In these conditions, the leaner `all-Pr` variant processes a larger tree with smaller per-node overhead and proves optimality for seven more instances.

7 Summary and Outlook

In this work, we propose an exact branch-and-price algorithm for job-shop scheduling with time-dependent operating costs and cardinality constraints. The method relies on an extended formulation over job schedules and on a pricing subproblem solved by a fast dynamic program. We build on this foundation with three components: a hierarchical branching scheme combining problem-specific ideas with MIP branching techniques; a propagation routine exploiting precedence, disjunctive, and capacity constraints; and primal heuristics that leverage this structure alongside inexpensive pricing calls to construct feasible solutions efficiently. Computational experiments on a large benchmark set show that the algorithm outperforms the compact time-indexed formulations in both feasibility and optimality. Beyond this empirical improvement, a key takeaway is that the integrality property of the pricing subproblem need not be a limitation in similar settings, as the LP relaxation is already strong, and the benefit of decomposition comes from computational leverage enabled by fast pricing.

On the algorithmic side, several enhancements fit naturally within the algorithm. These include combinatorial robust cuts, as well as refined branching schemes that incorporate more problem-specific information alongside other standard MIP techniques, in particular to address degeneracy. The fast pricing oracle also naturally leads to exploring procedures that rely on multiple oracle calls, such as the subgradient algorithm. Other objectives, such as the sum of weighted completion times, are accommodated directly through an appropriate choice of per-time-step costs in the pricing subproblem. More structural extensions, such as scheduling variants with alternative machine choices per operation (e.g., flexible job-shop), would require modest adaptations to the pricing subproblem and the propagation routine, while the master problem, the primal heuristics, and the core algorithmic framework carry over unchanged.

References

1. Achterberg, T., Berthold, T.: Hybrid branching. In: van Hoeve, W.J., Hooker, J.N. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 309–311. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
2. Artigues, C.: On the strength of time-indexed formulations for the resource-constrained project scheduling problem. Operations Research Letters (2017)
3. Benichou, M., Gauthier, J., Girodet, P., Hentges, G., Ribiere, G., Vincent, O.: Experiments in MILP. Mathematical Programming **1**(1), 76 – 94 (1971)
4. Berthold, T.: Measuring the impact of primal heuristics. Operations Research Letters **41**(6), 611–614 (2013)
5. Berthold, T.: RENS – The optimal rounding. Mathematical Programming Computation (2013)
6. Berthold, T., Lodi, A., Salvagnin, D.: Primal Heuristics in Integer Programming. Cambridge University Press (2025)

7. Bolusani, S., Besançon, M., Bestuzheva, K., Chmiela, A., Dionísio, J., Donkiewicz, T., van Doornmalen, J., Eifler, L., Ghannam, M., Gleixner, A., Graczyk, C., Halbig, K., Hedtke, I., Hoen, A., Hojny, C., van der Hulst, R., Kamp, D., Koch, T., Kofler, K., Lentz, J., Manns, J., Mexi, G., Mühmer, E., Pfetsch, M.E., Schlösser, F., Serrano, F., Shinano, Y., Turner, M., Vigerske, S., Weninger, D., Xu, L.: The SCIP Optimization Suite 9.0 (2024)
8. Desrosiers, J., Lübbecke, M., Desaulniers, G., Gauthier, J.B.: Branch-and-Price. Springer Cham (2026)
9. Drexl, A., Kimms, A.: Optimization guided lower and upper bounds for the resource investment problem. Journal of the Operational Research Society (2001)
10. Felloussi, M., Delorme, X., Gianessi, P.: A branch-and-cut algorithm for job-shop scheduling under time-of-use pricing and a peak power limit. European Journal of Operational Research (2026)
11. Gamrath, G.: Generic Branch-Cut-and-Price. Diploma thesis, TU Berlin (2007)
12. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976)
13. Hojny, C., Gally, T., Habeck, O., Lüthen, H., Matter, F., Pfetsch, M.E., Schmitt, A.: Knapsack polytopes: a survey. Annals of Operations Research **292**, 469–517 (2020)
14. Kolter, M., Grunow, M., Kolisch, R.: On branch-and-price for multi-project scheduling. In: 19th International Workshop on Project Management and Scheduling. pp. 69–72 (Apr 2024)
15. Lancia, G., Rinaldi, F., Serafini, P.: A time-indexed LP-based approach for min-sum job-shop problems. Annals of Operations Research **186**(1), 175–198 (2011)
16. Lübbecke, M.E.: Column generation. In: Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Ltd (2011)
17. Maher, S., Miltenberger, M., Pedroso, J.P., Rehfeldt, D., Schwarz, R., Serrano, F.: PySCIPOpt: Mathematical programming in Python with the SCIP Optimization Suite. In: Greuel, G.M., Koch, T., Paule, P., Sommese, A. (eds.) Mathematical Software – ICMS 2016. pp. 301–307. Springer International Publishing, Cham (2016)
18. Masmoudi, O., Delorme, X., Gianessi, P.: Job-shop scheduling problem with energy consideration. International Journal of Production Economics **216** (04 2019)
19. Mehrotra, A., Trick, M.A.: A column generation approach for graph coloring. INFORMS J. Comput. **8**(4), 344–354 (1996)
20. Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F.: Automation and combination of linear-programming based stabilization techniques in column generation. INFORMS J. Comput. **30**(2), 339–360 (May 2018)
21. Pessoa, A., Uchoa, E., de Aragão, M.P., Rodrigues, R.: Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. Mathematical Programming Computation **2**(3), 259–290 (Dec 2010)
22. Zhang, H., Zhao, F., Fang, K., Sutherland, J.W.: Energy-conscious flow shop scheduling under time-of-use electricity tariffs. CIRP Annals **63**(1), 37–40 (2014)