Brief Announcement: New Pruning Rules for Optimal Task Scheduling on Identical Parallel Machines

Matthew Akram
Dominik Schreiber
uqozz@student.kit.edu
dominik.schreiber@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Baden-Württemberg, Germany

ABSTRACT

We address optimal makespan-minimizing identical parallel machine scheduling ($P||C_{\max}$) by introducing new pruning rules for branch-and-bound (BnB) and integrating them into a prior BnB algorithm. Experimental results indicate that the presented rules are inexpensive to evaluate, applicable frequently, and extremely beneficial to the BnB algorithm's overall performance.

CCS CONCEPTS

• **Theory of computation → Branch-and-bound**.

KEYWORDS

scheduling, branch and bound, combinatorial optimization

ACM Reference Format:
Matthew Akram and Dominik Schreiber. 2024. Brief Announcement: New Pruning Rules for Optimal Task Scheduling on Identical Parallel Machines. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2024), June 17–21, 2024, Nantes, France.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3626183.3660268

1 INTRODUCTION

In this work we address unconstrained task scheduling on identical parallel machines. This problem is often denoted by its Graham notation $P||C_{\max}$ [5]—a three-part definition where P represents identical parallel machines, no side constraints are present, and $C_{\max}$ is the maximum makespan (completion time) of any machine, which we wish to minimize. Since this problem is strongly NP-hard [3], it is necessary to consider heuristic approaches and/or approximation schemes [1]. We are concerned with finding exact solutions and thus utilize a heuristic search method named *branch-and-bound* (BnB), which proved to be highly effective for this task [8]. Our main contributions are new rules for $P||C_{\max}$ which help to prune the search space of BnB.

Formally, a $P||C_{\max}$ *optimization instance* (W, m) is defined by n durations $W = \{w_1, \ldots, w_n\}$ of n corresponding jobs $J = \{j_1, \ldots, j_n\}$ and by the number m of identical processors $P = \{p_1, \ldots, p_m\}$. An instance (W, m) asks for an *assignment* $A = \{a_1, \ldots, a_n\}$ of jobs to machines ($1 \leq a_i \leq m$) such that the *maximum completion time* $C_{\max} := \max_i\{\sum_{k \mid a_k=i} w_k\}$ is minimized. By contrast, a *decision instance* (W, m, U) additionally imposes an *upper bound* U for $C_{\max}$ and poses the question whether a feasible solution exists. Throughout this paper we assume that the jobs are sorted by duration in decreasing order, i.e., $(w_1 \geq w_2 \geq \ldots \geq w_n)$.

A BnB algorithm for $P||C_{\max}$ is a tree-like search where we successively extend an initially empty *partial assignment* A of jobs to processors until $|A| = n$. At each *decision level* ℓ, where $|A| = \ell$, we identify a set of decisions ("branches") which each extend A by one element. In addition, we maintain admissible bounds on $C_{\max}$ during search, which allows us to exclude decisions which inevitably lead to sub-optimal solutions ("bound"). We recursively and heuristically search the remaining decisions. For each decision level ℓ, we define the *assigned workload* of processor p_x as $C_x^\ell := \sum_{i \mid a_i=x} w_i$. The *least loaded* processor is the one with smallest C_x^ℓ.

2 PRUNING RULES

In current research there are a few BnB based approaches to find exact solutions to $P||C_{\max}$ (see [8]). Since the search tree of possible assignments grows exponentially, it is important to find rules with which large branches can be left unexplored. Consequently, we restrict the search space by imposing further rules on the structure of the solution, while preserving at least one optimal assignment. We refer to these rules as *pruning rules*. Other common names include *dominance criteria* or (in some cases) *symmetry breaking*.

2.1 Prior Pruning Rules

Dell'Amico and Martello [2] present a number of BnB pruning rules. Note that any pruning rule for the $P||C_{\max}$ optimization problem is also a valid pruning rule for the $P||C_{\max}$ decision problem. For each rule we assume a partial assignment A at decision level ℓ.

Pruning Rule 1. *If there are processors p_i, p_k $(i < k)$ where $C_i^\ell = C_k^\ell$, then w.l.o.g. each assignment to p_k can be pruned.*

Pruning Rule 2. *If $\ell = n - 3$, only two options must be considered:*

1. *Assign each of the three final jobs to the least loaded processor respectively.*
2. *Assign the third-to-last job to the* second *least loaded processor, then assign the other two jobs as in* (1).

Pruning Rule 3. *If $i < m$ jobs remain unassigned, then only the i least loaded processors must be considered for the next decision.*

For more rules, we refer to the literature on the subject [2, 7, 8].

SPAA 2024, June 17–21, 2024, Nantes, France.

ACM ISBN 979-8-4007-0416-1/24/06
https://doi.org/10.1145/3626183.3660268

2.2 New Pruning Rules

When restricting ourselves to the decision problem, we can derive stronger pruning rules. Consider an instance (W, m, U) of the $P||C_{\max}$ decision problem where we already assigned $u < U$ work to a certain processor and now decide whether to assign j_i to this processor as well. Given the set $J_i = \{j_i, \ldots, j_n\}$ of *smaller jobs*, we can calculate the function

$$\phi(j_i, u) = \{J' \subseteq J_i \mid u + \sum_{j_k \in J'} w_k \leq U\}.$$

ϕ lists all possible combinations of jobs we can assign to a processor to still have an assigned workload (on this processor) that is at most U. With that we obtain the following pruning rules.

Pruning Rule 4. *Given an instance (W, m, U) of the decision problem of $P||C_{\max}$ and a partial assignment A with $|A| = \ell$, if there are processors p_x, p_y with assigned workload C_x^ℓ and C_y^ℓ such that $\phi(j_i, C_x^\ell) = \phi(j_i, C_y^\ell)$, then w.l.o.g. we can prune the decision $a_i = y$.*

This is a generalization of Pruning Rule 1. The reasoning behind this rule is that if two processors have equal ϕ sets for a given partial assignment, then any completion of one processor can also be used as a completion of the other. Therefore, this rule breaks symmetry and preserves feasibility of a given $P||C_{\max}$ decision instance.

Pruning Rule 5 (The Fill-Up Rule, *FUR*). *Given an instance (W, m, U) of the decision problem of $P||C_{\max}$ and a partial assignment A with $|A| = \ell$, let j_i be the largest unassigned job that can still be assigned to processor x (i.e., $C_x^\ell + w_i \leq U$). If*

$$w_i = \max\left\{ \sum_{j_k \in J'} w_k \mid J' \in \phi(j_i, C_x^\ell) \right\}$$

then w.l.o.g. we only need to consider the assignment $a_i = x$.

This rule is valid since any set of jobs that can be used to complete this processor will have a total weight of at most w_i. Thus, the jobs assigned to complete this processor can always be swapped with j_i in any completion where it is not assigned to this processor. We can check if this property holds by checking if $\phi(j_i, C_x^\ell) = \phi(j_i, U - w_i)$.

2.3 Efficient Computation

Since calculating ϕ explicitly would be prohibitively expensive, we introduce the auxiliary *range equivalency table (RET)*, which allows us to implicitly determine when $\phi(j_i, C_x^\ell) = \phi(j_i, C_y^\ell)$.

For the construction, we highlight two interesting properties of ϕ. Intuitively, one can see that $\phi(j_i, u) \supseteq \phi(j_i, u + 1)$ for all $u < U$. This is because the sets of admissible jobs will never decrease when increasing the available processing time. The second important property is the fact that for $i < n$

$$\phi(j_i, u) = \phi(j_{i+1}, u) \cup \{j_i \cup X \mid X \in \phi(j_{i+1}, u + w_i)\}. \qquad (1)$$

In words, the valid ways to assign jobs that may or may not include j_i is equal to the union of valid ways of assigning jobs that do not include j_i, and the valid ways of assigning jobs that do.

The RET is an $n \times (U + 1)$ table with entries in $\mathbb{N}$. The first dimension of the RET is indexed from 1 to n, since each row of the RET represents the equivalence ranges for a single job. The second dimension of the RET is indexed from 0 to U—one entry for each possible assigned workload. For a job j_i, an *equivalence range* is a range of workloads $u, \ldots, u'$ such that $RET[i][u] = \ldots = RET[i][u']$. For such a range we assert that $\phi(j_i, u) = \ldots = \phi(j_i, u')$.

We construct the RET going from the smallest (j_n) to the largest job (j_1). For j_n, $\phi(j_n, u) = \{\emptyset, \{j_n\}\}$ if $u + w_n \leq U$, and $\phi(j_n, u) = \{\emptyset\}$ otherwise. We thus initialize two equivalence ranges: $RET[n][U - w_n + 1] = \ldots = RET[n][U] = 1$ and $RET[n][0] = \ldots = RET[n][U - w_n] = 2$. For job j_i $(i < n)$, we denote the two entries of j_{i+1} that are relevant for applying property (1) as $\mathit{left}(i, u) := RET[i + 1][u]$ and $\mathit{right}(i, u) := RET[i + 1][u + w_i]$ (with $\mathit{right}(i, u) := 0$ if $u + w_i > U$). We start by setting $RET[i][U] := 1$ and then proceed sequentially for $u = U - 1, U - 2, \ldots, 0$:

$$RET[i][u] := \begin{cases} RET[i][u + 1], & \text{if } \mathit{left}(i, u) = \mathit{left}(i, u + 1) \\ & \wedge\ \mathit{right}(i, u) = \mathit{right}(i, u + 1), \\ RET[i][u + 1] + 1, & \text{otherwise.} \end{cases}$$

In words, we initialize a new equivalency range for j_i if *left* or *right* changes from $u + 1$ to u, and we extend the prior range otherwise.

Theorem 2.1. *Given an instance (W, m, U) of the $P||C_{\max}$ decision problem, for any $i \in \{1, \ldots, n\}$ and $u, u' < U$*

$$RET[i][u] = RET[i][u'] \iff \phi(j_i, u) = \phi(j_i, u').$$

The reasoning for this theorem is that the value of a *RET* entry changes iff $\phi(j_{i+1}, u) \neq \phi(j_{i+1}, u + 1)$ or $\phi(j_{i+1}, u + w_i) \neq \phi(j_{i+1}, u + w_i + 1)$. We provide a full proof online (see footnote 2).

The space requirements of the RET are in $O(U \cdot n)$. Note that we can compress this further to $O(U)$ by only storing at index u the index of the smallest job i where $RET[i][u] \neq RET[i][u + 1]$. While we do not discuss this technique in detail, we found it to significantly cut our scheduler's memory usage for large instances.

3 BRANCH AND BOUND ALGORITHM

We now present an adaptation of the algorithm by Dell'Amico and Martello [2] to demonstrate how our new pruning rules can be integrated into BnB algorithms for the $P||C_{\max}$ optimization problem (see also [8]). We begin with initializing upper and lower bounds U, L for the given instance. For this paper, we use the trivial lower bound [6] and the upper bound obtained using LPT [4]. Next, we set $U \longleftarrow U - 1$ and consider the decision problem $I = (W, m, U)$.

We then call Algorithm 1 with $A = \emptyset$. This recursive algorithm returns TRUE iff it found an improved solution. It proceeds as follows: We first perform some infeasibility checks given A (l. 1) and apply Rule 2 if only three unassigned jobs remain (l. 2–5).[1] Otherwise, we check if the FUR can be applied, and if so we apply it and recurse (l. 6–8). If that recursion is unsuccessful, then the respective bound cannot be improved using the current partial assignment and we return with the current best solution (l. 11). Otherwise, we assume U and the *RET* have been updated accordingly. We undo the assignment made by FUR and need to recurse again in case further improvements are possible given the new bound (l. 9–10). Lastly, if all else fails, we branch over all processors (l. 12–18) while using Rules 3 and 4. Note that the modifications made by a recursion may change which pruning rules apply to later processors, which is why we perform pruning individually and just-in-time.

[1] In this algorithm we assume w.l.o.g. that $n \geq 3$. The problem is trivial if $n < 3$.

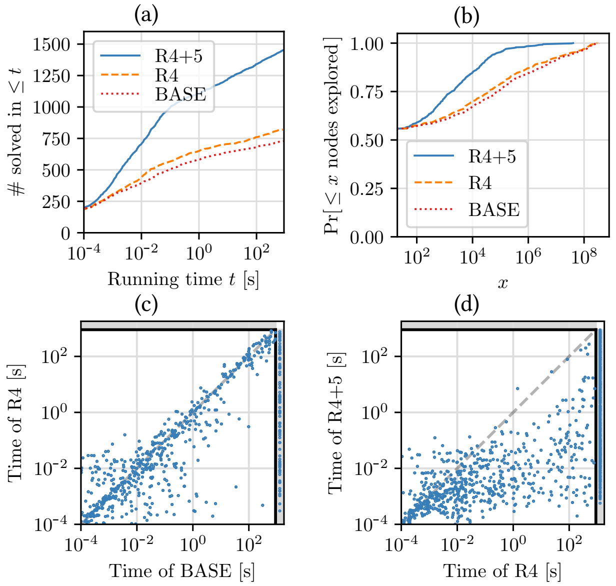

Figure 1: Solved instances relative to running time (a); distribution over explored nodes on commonly solved instances (b); scatter plot for BASE vs. R4 (c) and R4 vs. R4+5 (d).

Algorithm 1: ImproveSolution

```
Global Data: $W, m, RET, L, U$, BestSolution $= \bot$
Data: Partial Assignment $A$
if $Infeasible(A)$ then return false;
if $|A| = n - 3$ then
    BestSolution, $U \leftarrow$ Apply Rule 2;
    $RET \leftarrow RET :: new(W, m, U)$;
    return true;
if $\exists\, i, x : RET[i][U - w_i] = RET[i][C_x^{|A|}]$ then
    $A' \leftarrow A \cup \{a_i = x\}$ ;                // Apply FUR
    if $ImproveSolution(A')$ then
        $ImproveSolution(A)$ ;       // retry with new $U$
        return true;
    else return false;                        // infeasible!
$j_i \leftarrow$ largest unassigned job;  $improved \leftarrow$ false;
for each processor $p_x$ by free space descendingly do
    if $a_i = x$ can be pruned via Rules 3–4 then continue;
    $A' \leftarrow A \cup \{a_i = x\}$;
    $improved \leftarrow improved \lor ImproveSolution(A')$;
end
return $improved$
```

4 EVALUATION

We now evaluate the effect of our pruning rules on Algorithm 1. We consider 3500 problem instances defined by Mrad and Souyah [9] with $n/m \in [2, 3]$, $n \in [20, 220]$, and both uniform and normal distributions to generate job sizes. We implemented our approach in Rust and run the experiments in parallel on a 128-core (256-thread) AMD EPYC 77132 server with 2 TB of DDR4 RAM.

We considered three runs: The baseline BnB algorithm without computing the RET ("BASE"), which uses Rules 1-3; an enhancement where we compute the RET and use it for Rule 4 ("R4") instead of Rule 1; and the full Algorithm 1 also including Rule 5 ("R4+5").

R4 was able to solve 92 more problems (+12.6%) than BASE and R4+5 was able to solve 724 more problems (+99%) than BASE (see Fig. 1a). Rule 4 thus appears to bring modest improvements while the FUR results in substantial performance benefits. In line with this observation, the number of explored nodes per problem (Fig. 1b) decrease by a geometric mean factor of 1.78 from BASE to R4 and 36.9 from R4 to R4+5 (49.4 from BASE to R4+5). When enabling FUR, scheduling times decrease consistently and, in many cases, by several orders of magnitude (Fig. 1d). The optimal makespans reported by R4+5 ranged from 90 to 1503 (median 163, mean 217).

We provide a comparison to an ILP-based approach [9] online.[2] In short, our approach uses much less memory and outperforms ILP on a situational basis. Adding more advanced bounding techniques may render our BnB algorithm even more competitive.

5 CONCLUSION

We introduce new pruning rules for BnB-based optimal $P||C_{\max}$ scheduling which empirically result in substantial performance improvements to a prior BnB algorithm. In the future, we are interested in combining our rules with other promising $P||C_{\max}$ scheduling techniques such as generic translation or randomized search.

ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882500).

REFERENCES

[1] Sebastian Berndt, Max A. Deppert, Klaus Jansen, and Lars Rohwedder. 2022. Load balancing: The long road from theory to practice. In *Proc. ALENEX*. 104–116. https://doi.org/10.1137/1.9781611977042.9

[2] Mauro Dell'Amico and Silvano Martello. 1995. Optimal Scheduling of Tasks on Identical Parallel Processors. *INFORMS J. Computing* 7 (05 1995), 191–200. https://doi.org/10.1287/ijoc.7.2.191

[3] Michael R. Garey and David S. Johnson. 1978. Strong NP-completeness results. *J. ACM* 25, 3 (1978), 499–508. https://doi.org/10.1145/322077.322090

[4] Ronald L. Graham. 1966. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.* 45 (1966), 1563–1581. https://doi.org/10.1002/j.1538-7305.1966.tb01709.x

[5] Ronald L. Graham, Eugene L. Lawler, Jan K. Lenstra, and AHG Rinnooy Kan. 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*. Vol. 5. 287–326. https://doi.org/10.1016/s0167-5060(08)70356-x

[6] Mohamed Haouari, Anis Gharbi, and Mahdi Jemmali. 2006. Tight bounds for the identical parallel machine scheduling problem. *Int. Trans. Operational Research* 13 (10 2006), 529 – 548. https://doi.org/10.1111/j.1475-3995.2006.00562.x

[7] Mohamed Haouari and Mahdi Jemmali. 2008. Tight bounds for the identical parallel machine-scheduling problem: Part II. *Int. Trans. Operational Research* 15, 1 (2008), 19–34. https://doi.org/10.1111/j.1475-3995.2007.00605.x

[8] Alexander Lawrinenko. 2017. *Identical parallel machine scheduling problems: structural patterns, bounding techniques and solution procedures.* Ph. D. Dissertation. Jena. https://www.db-thueringen.de/receive/dbt_mods_00032188

[9] Mehdi Mrad and Nizar Souayah. 2018. An Arc-Flow Model for the Makespan Minimization Problem on Identical Parallel Machines. *IEEE Access* 6 (2018), 5300–5307. https://doi.org/10.1109/ACCESS.2018.2789678

[2] https://github.com/matthewakram/spaa24-pcmax-bnb.git