Low-Cost Algorithms for the Restricted Assignment Problem on Intervals of Machines

Louis-Claude Canon
FEMTO-ST, Univ. Franche-Comté
Besançon, France
louis-claude.canon@femto-st.fr

Anthony Dugois
LIP, ENS Lyon, Inria
Lyon, France
anthony.dugois@ens-lyon.fr

Loris Marchal
LIP, ENS Lyon, CNRS, Inria
Lyon, France
loris.marchal@ens-lyon.fr

ABSTRACT

We consider a special case of the Restricted Assignment problem where processing sets of jobs consist in contiguous intervals of machines. This problem is known as the Restricted Assignment problem on Intervals (RAI), and arises, for instance, when partitioning *multi-get* requests in distributed and replicated NoSQL databases. These databases aim at providing high availability and very low latency, i.e., the algorithms used to schedule requests must have good execution time. In this paper, we propose optimal and approximation algorithms for some variants of the RAI problem that apply to these systems. On the basis of previous work done by Lin et al. (*European Journal of Operational Research*, 2004), we present an efficient $(2 - 1/m)$-approximation algorithm, where m is the number of machines. We also provide a generalization of the RAI problem on so-called *circular* intervals, i.e., intervals that appear when machines are arranged in a circular fashion, and we call this new problem the Restricted Assignment problem on Circular Intervals (RACI). We give a procedure that computes an optimal solution when there are K possible processing times, assuming that one already knows an optimal algorithm for the corresponding non-circular problem. This enables us, for instance, to obtain an optimal algorithm with complexity $O(n \log n)$ for the RACI problem with a fixed number of machines and unitary jobs.

KEYWORDS

Scheduling, Makespan, Restricted Assignment, Intervals, Approximation, Multi-get

ACM Reference Format:
Louis-Claude Canon, Anthony Dugois, and Loris Marchal. 2023. Low-Cost Algorithms for the Restricted Assignment Problem on Intervals of Machines. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Many theoretical scheduling problems capture the essence of practical challenges in modern distributed systems. Among those, NoSQL databases such as distributed key-value stores, which spread data over several servers and map items to unique keys, became central components in the architecture of online cloud applications, thanks to their excellent performance and capability to scale linearly with the dataset. They are often subject to high throughputs, and must therefore be able to serve requests with low-latency to meet user expectations. Hence, the proper scheduling of these requests is of paramount importance, and has a direct effect on the overall observed performance of the system. The API of modern distributed key-value stores offer various operations to interact with the dataset, among which single reads and writes are the most common. However, as most webservices often need to retrieve several data items to perform their own calculations, some APIs provide a special type of operations called *multi-get* requests, which permit to retrieve several items from a given keyset in a single round-trip. When executing such a multi-get request, the key-value store needs to partition the keyset into several suboperations at the destination of storage servers, and it should balance the keys between these suboperations in order to respond as quickly as possible.

To ensure accessibility of data in case of node failure, which happens regularly, the dataset is usually replicated on several servers. This makes each single key accessible at different locations. Therefore, the multi-get request scheduling process is quasi-identical to the so-called Restricted Assignment problem, which is a well-known problem whose objective is to schedule jobs to machines in such a way that the makespan is minimized, with the additional constraint that a given job can be processed by a subset of machines only. As this problem is NP-hard, partitioning multi-get requests in an optimal manner clearly cannot be done in reasonable time in the general case. On the positive side, the actual variant of the Restricted Assignment problem that applies to multi-get request partitioning is slightly easier than the general problem: the data replication strategy often consists in duplicating items in such a way that the processing sets are contiguous intervals of machines. This enables deriving guaranteed algorithms for multi-get partitioning, with the additional challenge that these algorithms must run as quickly as possible, as the time taken to compute a sufficiently good solution must not become the bottleneck in the request service.

Contributions. We start from prior work, done by Lin et al. [11], on the Restricted Assignment problem on Intervals (RAI). The authors provide an efficient algorithm in the special case of unitary jobs, although we find their analysis to be incorrect. We take their results further by generalizing the proposed algorithm, providing a corrected proof and a corrected version of its complexity analysis. In the case of unitary jobs, this enables us to derive an optimal algorithm, called Estimated Least Flexible Job (ELFJ), which runs in time $O(m^2 + n \log n + mn)$, where m is the number of machines and n is the number of jobs. We also prove that ELFJ (with a small adaptation) is a $(2-1/m)$-approximation algorithm when jobs have arbitrary processing times.

Moreover, we further generalize the RAI problem by introducing the notion of *circular* intervals, i.e., intervals that may begin at the end of the list of machines and loop back to the start, which match the actual replication strategy of distributed key-value stores. We call this generalization the Restricted Assignment problem on Circular Intervals (RACI). In the case of K different processing times, we develop a general framework that provides an optimal algorithm for the RACI problem, running in time $O(n^K f(n))$, assuming that one knows a polynomial algorithm running in time $O(f(n))$ for the corresponding non-circular problem. We illustrate how this framework applies to existing solutions by giving an optimal algorithm for the RACI problem with K job types, running in time $O(mn^{3K} \log \sum p_j)$. Finally, we revisit the case with unitary jobs by adapting ELFJ for the RACI version, and we show that this generalization does not affect the final complexity of the algorithm, i.e., it also runs in time $O(m^2 + n \log n + mn)$.

We present the model in Section 2, where we formally define the Restricted Assignment problem and its variants, and where we explain the motivation of this work more exhaustively. We review related work in Section 3. In Section 4, we present ELFJ, which is a generalization of the algorithm proposed by Lin et al. [11] for the RAI problem on unitary jobs, and we develop our general framework for the circular problem in Section 5. We conclude the paper in Section 6.

2 MODEL & APPLICATIVE CONTEXT

We present the formal model in this section, and we introduce the theoretical problem that is at the basis of a practical challenge in distributed key-value stores, namely, the optimal partitioning of multi-get requests.

2.1 The Restricted Assignment Problem

In the problem of scheduling jobs on unrelated machines (also known as the $R||C_{\max}$ problem), we are given a set of n jobs $J = \{1, \ldots, n\}$ and a set of m machines $M = \{1, \ldots, m\}$, where each job $j \in J$ has processing time $p_{ij} > 0$ on machine $i \in M$. The objective is to schedule (non-preemptively) the jobs on machines so as to minimize the makespan, that is to say, the maximum completion time of the jobs.

The Restricted Assignment (RA) problem is a special case of $R||C_{\max}$, where each job $j \in J$ can be processed only on a subset of machines $\mathcal{M}_j \subseteq M$, which we call the *processing set* of j. In this setting, the job j has processing time p_j on machine i if and only if $i \in \mathcal{M}_j$, and $+\infty$ otherwise. The problem is often noted $P|\mathcal{M}_j|C_{\max}$ in Graham's notation. A given instance of the RA problem can be represented by a bipartite graph $G = (J \cup M, E)$, where there is an edge (j, i) with weight p_j between a job $j \in J$ and a machine $i \in M$ if and only if $i \in \mathcal{M}_j$.

The $R||C_{\max}$ problem, and more specifically the RA problem, are well-known NP-hard problems, and it has even been proved that no algorithm can approximate an optimal solution within a factor better than $3/2$ unless P = NP [10]. Hence, specific cases of the RA problem have also been the subject of extensive research. One possible manner to reduce the complexity of the problem is to bring structure in the processing sets of jobs. For example, a common subproblem consists in solving the RA problem on *nested*[1] processing sets, that is to say, one of the following properties holds for any two jobs $j, j' \in J$: $\mathcal{M}_j \subseteq \mathcal{M}_{j'}$, $\mathcal{M}_{j'} \subseteq \mathcal{M}_j$, or $\mathcal{M}_j \cap \mathcal{M}_{j'} = \emptyset$. An even more specific case is when the processing sets are *inclusive*, i.e., for any two jobs $j, j' \in J$, either $\mathcal{M}_j \subseteq \mathcal{M}_{j'}$, or $\mathcal{M}_{j'} \subseteq \mathcal{M}_j$.

In this paper, we focus on *interval* processing sets, that is a subcase of the RA problem, but more general than the nested case. In this problem, the machines can be rearranged such that the processing sets of jobs consist in contiguous intervals of machines. More formally, let us note (a, b) the interval ranging from machine a (inclusive) to machine b (inclusive, $a \leq b$), and $I_{(a,b)} = \{a, a+1, \ldots, b\}$. In the Restricted Assignment problem on Intervals (RAI), for all jobs $j \in J$, we define $\mathcal{M}_j = I_{(a_j, b_j)}$, where a_j and b_j are respectively the lower and upper bounds of the interval of machines on which the job j can be assigned. In this case, an instance of the problem can be seen as a *convex* bipartite graph.

2.2 Multi-Get Requests in Storage Systems

An applicative context in which the RAI problem arises is when partitioning multi-get requests in a distributed and replicated storage system. Key-value stores are low-latency databases where each data item is associated with a unique key, and a simple read operation consists in retrieving the value that corresponds to a given key [2, 9]. These databases are distributed on several nodes, as the dataset is often too large to fit on a single server. Thus, the dataset is split in several partitions, and each partition is stored on a different server. Moreover, in order to guarantee accessibility of data in case of node failure, each partition is replicated several times on different servers, and the duplicated data on a given node form a *replica* of the original partition. The replication strategy differs from one system to another, but a common and efficient way consists in arranging the servers on a virtual ring, and replicating the partition of each server i on its two successors $i+1$ and $i+2$ (modulo the number of servers). In other words, servers are virtually ordered, and each key/value couple is stored on an interval of three different consecutive servers.

Multi-get requests are read operations that involve several keys. These aggregated operations are useful, for instance, to reduce the number of network round-trips between the database and a webserver, where a single HTTP request often requires to retrieve several data items before responding to the client [6, 13]. In such a multi-get request, the keys (which are called the *keyset* of the request) may be located in different partitions. Thus, the system must split the request into several subrequests by partitioning the keyset into several *opsets*, and redirect each opset toward the appropriate server (which must hold the keys of the opset, as a replica or not). As we cannot respond before executing all opsets, these opsets must be balanced to guarantee a good response time for the overall multi-get request, i.e., we do not want a few very fast opsets and one that is very slow.

This opset partitioning problem can be seen as the RAI problem, where servers are the machines, and each read operation for a given key is a job, whose processing time is the time required to retrieve the corresponding data item (which depends on the number of

[1]This special case is also known as *laminar* processing sets.

bytes that represent the stored item), and the processing set is the interval of servers on which the key is located. An opset is the set of keys whose corresponding jobs are assigned to the same machine. As we seek to minimize imbalance between opsets, we want to minimize the maximum completion time of each opset, which is the makespan. Of course, when partitioning a multi-get request, one can also take into account the current load of each machine by adding m fake jobs whose processing set consists of only one machine.

As the number of jobs and machines are relatively small in this context, a simple solution to this problem could consist in an integer programming formulation, where c is the makespan, and each x_{ij} is a binary variable that indicates whether the job j is assigned to the machine i:

$$\begin{aligned} \textbf{minimize} \quad & c \\ \text{subject to} \quad & \forall j \in J, \sum_{i \in M} x_{ij} = 1, \\ & \forall i \in M, \sum_{j \in J} x_{ij} p_j \leq c, \\ & x_{ij} \in \{0, 1\}. \end{aligned}$$

However, we must solve this problem for each multi-get request in real time, and key-value store systems are usually dimensioned to handle high throughputs (of the order of thousands of requests per second). This approach is, therefore, clearly not scalable. We need a polynomial, guaranteed (even if not optimal), greedy algorithm, to ensure that the time taken to partition a multi-get request is not greater than the time required to execute the request itself.

Among the interesting, potentially simpler, variants of the RAI problem, we can mention the case with unitary jobs. This corresponds to near-identical requests in the key-value store, which may happen if all data items have similar sizes. Another variant is the case with K types of jobs, which corresponds to the case where we can categorize requests, for example by considering *small* and *large* data items.

3 RELATED WORK

The Restricted Assignment problem has received a significant attention, as it captures the essence of many practical problems. It is a subcase of the more general Unrelated Scheduling problem ($R||C_{\max}$), for which a famous 2-approximation algorithm, based on linear programming, has been proposed by Lenstra et al. [10]. The authors also considered the RA problem and proved that no polynomial algorithm may give an approximation better than 3/2, unless P = NP. A Quasi-Polynomial Time Approximation Scheme (QPTAS) has recently been derived for the RA problem, which approximates an optimal solution within a factor $11/6 + \varepsilon$ in time $O((n + m)^{O(1/\varepsilon \log(n+m))})$ [8].

Various subcases of the RA problem have also been considered in the literature. For instance, Ebenlendr et al. [3] studied the Graph Balancing problem, which corresponds to the RA problem where each job may be processed by at most two different machines. They show the 3/2-hardness of the problem, and provide a 7/4-approximation algorithm. List-scheduling is a famous $(2 - 1/m)$-approximation for the problem $P||C_{\max}$, and it has also been proved to give the same guarantee for the nested case of RA, at the condition that jobs are initially sorted by non-decreasing size of their processing set [5]. The authors also derived an efficient 3/2-approximation for the inclusive case, running in time $O(nm \log m)$. There also exist some results focused on Polynomial Time Approximation Scheme (PTAS). On the negative side, Maack et al. proved that no PTAS exists for the interval case of RA (RAI) unless P = NP [12]. Interestingly, there is a PTAS when all intervals are overlapping without any strict inclusion, i.e., for any two jobs j, j', $\mathcal{M}_j \not\subset \mathcal{M}_{j'}$ and $\mathcal{M}_{j'} \not\subset \mathcal{M}_j$ [14]. There is also a PTAS for the nested case and other variants [4].

Some authors also studied the RA problem with restricted processing times. Jansen et al. proved that even when considering only two possible processing times, there is no algorithm giving an approximation better than 4/3 [7]. However, in the specific case where $p_j \in \{1, 2\}$, there is a 3/2-approximation algorithm [5]. By approaching the RA problem as a matching problem, Biró et al. derive a $(2 - 1/2^k)$-approximation when $p_j \in \left\{1, 2, \ldots, 2^k\right\}$ for all jobs [1]. They also give an optimal algorithm when the same constraint is applied on the nested case. When jobs are unitary, the RA problem becomes simpler, and it is possible to find optimal schedules in time $O(n^3 \log n)$ by coupling a binary search procedure to a network flow formulation of the problem [11].

4 AN ALGORITHM FOR REGULAR INTERVALS

Let us simplify the practical problem for now by considering only regular intervals of machines, i.e., we focus on the standard RAI problem. Lin et al. [11] proposed a polynomial algorithm to solve the RAI problem when jobs are unitary. They argue that their algorithm runs in time $O(m(m + n))$, although we found their analysis to be slightly incorrect. We also noticed an error in their proof of optimality. In this section, we give a correct version of their proof, and we generalize their approach to derive the following results:

(i) an optimal algorithm for the RAI problem with unitary jobs, which runs in time $O(m^2 + n \log n + mn)$ (Theorem 1), and
(ii) a tight $(2 - 1/m)$-approximation algorithm for the RAI problem with arbitrary jobs, which also runs in time $O(m^2 + n \log n + mn)$ (Theorem 2).

Let us introduce Algorithm 1, called Estimated Least Flexible Job (ELFJ), which is directly inspired by Lin et al.'s algorithm. ELFJ takes a parameter λ and builds a schedule that is guaranteed to finish before time λ. In other words, λ denotes an upper bound on the optimal makespan: as λ gets closer to the actual optimal makespan, ELFJ gets closer to an optimal schedule. ELFJ performs two steps: first, it sorts the jobs in non-decreasing order of interval upper bound b_j (in time $O(n \log n)$), and then it assigns jobs on the machines (in time $O(mn)$).

Before starting the optimality analysis, let us introduce some notations and definitions. For any interval of machines (α, β), where $1 \leq \alpha \leq \beta \leq m$, we define $K_{(\alpha,\beta)}$ as the set of jobs whose processing set is included in $I_{(\alpha,\beta)}$, i.e., $K_{(\alpha,\beta)} = \left\{j \in J \text{ s.t. } \mathcal{M}_j \subseteq I_{(\alpha,\beta)}\right\}$. We denote the total processing time of jobs in $K_{(\alpha,\beta)}$ by $w_{(\alpha,\beta)}$, i.e., $w_{(\alpha,\beta)} = \sum_{j \in K_{(\alpha,\beta)}} p_j$. Let $\tilde{w}_{(\alpha,\beta)}$ represent the minimum average work that *any* schedule must perform on machines $\alpha, \dots, \beta$, i.e.,

$$\tilde{w}_{(\alpha,\beta)} = \frac{w_{(\alpha,\beta)}}{\beta - \alpha + 1},$$

and let $\tilde{w}_{\max}$ be the maximum value of $\tilde{w}_{(\alpha,\beta)}$ over all intervals ($\tilde{w}_{\max} = \max_{1 \le \alpha \le \beta \le m} \left\{\tilde{w}_{(\alpha,\beta)}\right\}$). From these definitions, we can easily derive a lower bound on the optimal makespan $C_{\max}^{OPT}$ for a given instance $\mathcal{I}$ of the RAI problem, as shown by Lin et al. in their original work [11].

Lemma 1. *The optimal makespan is bounded by* $\tilde{w}_{\max}$*, i.e.,* $C_{\max}^{OPT} \ge \tilde{w}_{\max}$.

Proof: Let $C_{(\alpha,\beta)}^{OPT}$ be the maximum completion time of machines $\alpha, \dots, \beta$ in an optimal schedule. We clearly have $C_{(\alpha,\beta)}^{OPT} \ge \tilde{w}_{(\alpha,\beta)}$ for any interval (α, β), because all jobs in the set $K_{(\alpha,\beta)}$ must be done between machines α and β. In the best case, the jobs are perfectly balanced on $\beta - \alpha + 1$ machines.

Let (a, b) be the interval of machines such that $\tilde{w}_{\max} = \tilde{w}_{(a,b)}$. Then we have $C_{(a,b)}^{OPT} \ge \tilde{w}_{(a,b)}$. Of course, $C_{\max}^{OPT} \ge C_{(a,b)}^{OPT}$, i.e., $C_{\max}^{OPT} \ge \tilde{w}_{\max}$. ■

Corollary 1. *If all processing times are integers, then* $C_{\max}^{OPT} \ge \lceil \tilde{w}_{\max} \rceil$.

Proof: Suppose that all processing times are integers in the considered instance. Then we have $C_{(\alpha,\beta)}^{OPT} \ge \lceil \tilde{w}_{(\alpha,\beta)} \rceil$ for any interval (α, β), because jobs are not divisible. The rest of the proof is analogous to the previous lemma. ■

4.1 Computing $\tilde{w}_{\max}$

The idea of Lin et al. is to use $\lceil \tilde{w}_{\max} \rceil$ as the value of the parameter λ in ELFJ to get an optimal schedule when jobs are unitary. This means that one must be able to compute $\tilde{w}_{\max}$ in polynomial time in order to apply the algorithm. Suppose for a moment that all processing times are unitary, i.e., for all interval (α, β), $w_{(\alpha,\beta)} = \left|K_{(\alpha,\beta)}\right|$. In the original paper, the authors propose the following procedure. First, for all machine i, construct the sets $A_i = \left\{j \in J \text{ s.t. } i \le a_j\right\}$ and $B_i = \left\{j \in J \text{ s.t. } b_j \le i\right\}$ in time $O(mn)$. Then, for all intervals (α, β), compute $\left|K_{(\alpha,\beta)}\right| = \left|A_\alpha \cap B_\beta\right|$. Counting the number of common elements in two sets is clearly not a constant-time operation. Hence, as there are $O(m^2)$ possible intervals, the time complexity of this procedure is at least $O(c(n) \cdot m^2)$, where $c(n)$ is the time complexity of counting common elements in two sets of size $O(n)$. If we recall that the original algorithm performs a sorting operation (in time $O(n \log n)$) and the assignment of jobs to machines (in time $O(mn)$), we conclude that the total complexity $O(m^2 + mn)$ given by Lin et al. is underestimated, and we argue that their approach gives in fact an algorithm with time complexity $O(c(n) \cdot m^2 + n \log n + mn)$. Moreover, their computation method is not suitable to the case where processing times are arbitrary.

We present in this section a new procedure to compute $\tilde{w}_{\max}$ in time $O(m^2 + n)$ for any instance of the RAI problem with arbitrary processing times. We notice that the set of intervals in a list of m machines can be represented by a graph, where nodes correspond to intervals. For all intervals (α, β) such that $\alpha < \beta$, the node (α, β) is the parent of two children nodes $(\alpha, \beta - 1)$ and $(\alpha + 1, \beta)$ (see Figure 1).

Let $J_{(\alpha,\beta)}$ be the set of jobs whose processing set is exactly $I_{(\alpha,\beta)}$, i.e., $J_{(\alpha,\beta)} = \left\{j \in J \text{ s.t. } \mathcal{M}_j = I_{(\alpha,\beta)}\right\}$, and let $v_{(\alpha,\beta)}$ be their total processing time. Then we have a recursive relation between the values $w_{(\alpha,\beta)}$: for a given interval (α, β) that has two children intervals, the work that must be done on machines $\alpha, \dots, \beta$ includes

(i) $J_{(\alpha,\beta)}$,
(ii) the work that must be done on machines $\alpha, \dots, \beta - 1$,
(iii) the work that must be done on machines $\alpha + 1, \dots, \beta$,
(iv) minus the work that must be done on machines $\alpha + 1, \dots, \beta - 1$, as it is included two times in (ii) and (iii).

Then, for any α, β, we have

$$w_{(\alpha,\beta)} = v_{(\alpha,\beta)} + w_{(\alpha,\beta-1)} + w_{(\alpha+1,\beta)} - w_{(\alpha+1,\beta-1)},$$

with the particular case $w_{(\alpha,\beta)} = 0$ if $\alpha > \beta$. The values $v_{(\alpha,\beta)}$ can be pre-computed in time $O(n)$ by scanning jobs, and the computation of the values $w_{(\alpha,\beta)}$ is done in time $O(m^2)$. Thus $\tilde{w}_{\max}$ can be found in time $O(m^2 + n)$ and space $O(m^2)$ (see Algorithm 2).

4.2 Optimality of ELFJ for Unitary Jobs

We now prove the optimality of ELFJ when all jobs are unitary and $\lambda = \lceil \tilde{w}_{\max} \rceil$. The principle of the proof comes from the work of Lin et al. [11], although we found their demonstration to be incorrect. We know from Corollary 1 that the optimal makespan is at least λ. Thus we seek to prove that ELFJ gives a schedule S whose makespan is at most λ.

By contradiction, Lin et al. assume there exists a unitary job j that could not be assigned by ELFJ on any machine before time

Algorithm 1 ELFJ

```
1: sort jobs in non-decreasing order of $b_j$
2: for each machine $i$ do
3:     $\rho_i \leftarrow 0$
4:     for each non-assigned job $j$ such that $a_j \le i \le b_j$ do
5:         if $\rho_i + p_j \le \lambda$ then
6:             assign $j$ to $i$
7:             $\rho_i \leftarrow \rho_i + p_j$
```

Algorithm 2 Computing $\tilde{w}_{\max}$ in time $O(m^2 + n)$

```
1: $\tilde{w}_{\max} \leftarrow 0$
2: for each $0 \le \alpha \le \beta \le m$ do
3:     $v_{(\alpha,\beta)} \leftarrow 0$
4: for each job $j$ do
5:     $v_{(a_j,b_j)} \leftarrow v_{(a_j,b_j)} + p_j$
6: for all $l$ from 0 to $m - 1$ do
7:     for all $a$ from 1 to $m - l$ do
8:         $b \leftarrow a + l$
9:         $w_{(a,b)} \leftarrow v_{(a,b)} + w_{(a,b-1)} + w_{(a+1,b)} - w_{(a+1,b-1)}$
10:        $\tilde{w}_{(a,b)} \leftarrow \frac{w_{(a,b)}}{b-a+1}$
11:        if $\tilde{w}_{(a,b)} > \tilde{w}_{\max}$ then
12:            $\tilde{w}_{\max} \leftarrow \tilde{w}_{(a,b)}$
```

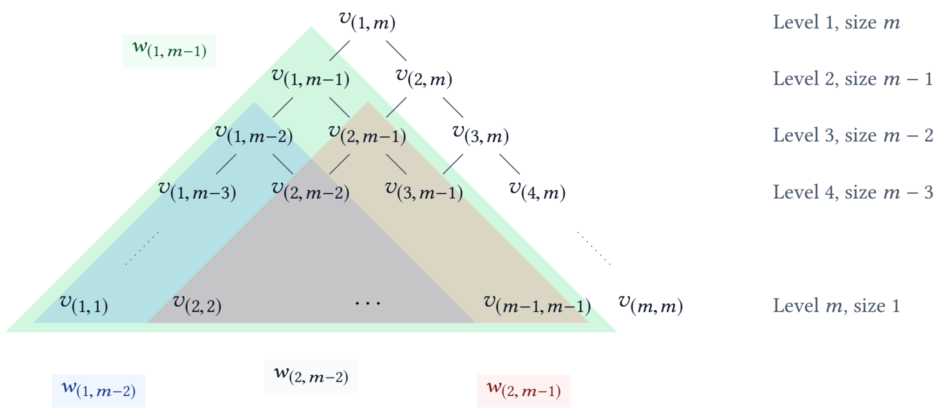

Figure 1: Interval hierarchy represented as a lattice graph. Each node represents an interval (x, y) and is labeled with the value $v_{(x,y)}$. Nodes are organized by level, where nodes on level h represent intervals of size $m - h + 1$, e.g., if $m = 3$, the node on level 1 is the interval of size 3, nodes on level 2 are intervals of size 2, and nodes on level 3 are intervals of size 1.

λ, which means that all machines between a_j and b_j must be full. Then we consider the machine with smallest index $\alpha \le a_j$ such that all machines between α and b_j are full. Let $\beta = b_j$. Now the goal is to prove that all jobs assigned by ELFJ on machines $\alpha, \alpha + 1, \ldots, \beta$ come from the set $K_{(\alpha,\beta)}$; in other words, the processing set of each job assigned on these machines is included in $I_{(\alpha,\beta)}$. Proving this property leads to the conclusion $\lambda < \tilde{w}_{(\alpha,\beta)}$, which is a contradiction because $\lambda = \lceil \tilde{w}_{\max} \rceil$.

To do so, Lin et al. argue that any job j' assigned on a machine between α and β must have $a_{j'} \ge \alpha$, otherwise j' would have been put on $\alpha - 1$ (which is not full), and $b_{j'} \le \beta$, because jobs have been assigned by non-decreasing order of b_j. This last justification is an error, as highlighted by the following counterexample: suppose that $\alpha = a_j - 1$, and there are λ jobs with interval $\alpha, \alpha + 1, \ldots, \beta + 1$ (call these jobs the *filling* jobs). The job j must be done in the interval $\alpha + 1, \alpha + 2, \ldots, \beta$. Then, the filling jobs will be assigned on machine α by ELFJ, even if we have $b_{j'} = \beta + 1 > \beta$ for all filling jobs j', because they are the only jobs that are feasible on α. Therefore, we cannot conclude that all jobs assigned on machines $\alpha, \alpha + 1, \ldots, \beta$ come from $K_{(\alpha,\beta)}$.

We present here a constructive proof that also consists in exhibiting a contradiction by finding a machine $\alpha \le a_j$ such that all jobs assigned between α and β come from $K_{(\alpha,\beta)}$. However, α is more carefully chosen in this new version: we start from the interval (a_j, b_j) and we extend this interval step by step until the appropriate condition is met.

Theorem 1. *Let $\lambda = \lceil \tilde{w}_{\max} \rceil$. Then ELFJ (Algorithm 1) is optimal and runs in time $O(m^2 + n \log n + mn)$.*

Proof: The beginning of the proof is similar to the one of Lin et al. By contradiction, suppose that ELFJ does not give a feasible schedule with makespan at most λ. Let j_0 be one of the non-assigned jobs. Then, as all jobs are unitary and λ is an integer, all machines in $\mathcal{M}_{j_0}$ must finish at least at time λ. Let $\beta = b_{j_0}$, and let $\gamma \le a_{j_0}$ be the smallest machine index such that all machines between γ and β complete at time λ. This means that the machine $\gamma - 1$ completes before time λ if $\gamma > 1$.

Now our goal is to find a machine α between γ and a_{j_0} such that all jobs assigned on machines $\alpha, \alpha + 1, \ldots, \beta$ come from the set $K_{(\alpha,\beta)}$. The process here is constructive. For the first step, let j be a job assigned on a machine between a_{j_0} and β. Then, we have $b_j \le \beta$, otherwise j_0 would have been scheduled instead of j. Now there are two cases: either we have $a_j \ge a_{j_0}$ for all j assigned between a_{j_0} and β, or $a_j < a_{j_0}$ for at least one job j assigned between a_{j_0} and β.

If the first case holds, then we set $\alpha = a_{j_0}$ and we are done: all jobs assigned between α and β have a processing set include in $I_{(\alpha,\beta)}$. If the second case holds, let us choose such j with the smallest a_j (then $a_j < a_{j_0}$), and let us call this job j_1. Now we proceed to the next step. If j_1 has been assigned between a_{j_0} and β, it means that $b_j \le b_{j_1} \le \beta$ for all jobs j assigned on machines $a_{j_1}, a_{j_1} + 1, \ldots, a_{j_0} - 1$, otherwise we would have scheduled j_1 instead. Moreover, we have two cases again: either we have $a_j \ge a_{j_1}$ for all j assigned between a_{j_1} and $a_{j_0} - 1$, or $a_j < a_{j_1}$ for at least one job j assigned between a_{j_1} and $a_{j_0} - 1$.

In the first case, we set $\alpha = a_{j_1}$ and we are done. Otherwise, we choose j with the smallest a_j, we call this job j_2 and we proceed to the next step by repeating the same reasoning.

To conclude, note that we have $a_j \ge \gamma$ for all jobs j assigned on a machine whose index is greater than or equal to γ, otherwise j would have been put on machine $\gamma - 1$, as it completes before time λ. By applying the described process iteratively, we inevitably reach a step k where there cannot exist a job j such that $a_j < a_{j_k}$, and we set $\alpha = a_{j_k}$.

Therefore, there exist $\alpha \le \beta$ such that

(i) $j_0 \in K_{(\alpha,\beta)}$,
(ii) machines $\alpha, \alpha + 1, \ldots, \beta$ complete at time λ, and
(iii) all jobs assigned on machines $\alpha, \alpha + 1, \ldots, \beta$ belong to $K_{(\alpha,\beta)}$.

Then we have

$$w_{(\alpha,\beta)} \ge (\beta - \alpha + 1)\lambda + 1 > (\beta - \alpha + 1)\lambda,$$

i.e., $\lambda < \tilde{w}_{(\alpha,\beta)}$, which is a contradiction. Hence, ELFJ gives a schedule feasible in λ time units, which means that $C_{\max}^{OPT} \le \lambda$. By Corollary 1, we also know that $C_{\max}^{OPT} \ge \lambda$. We conclude that

$C_{\max}^{OPT} = \lambda$, thus ELFJ is optimal. Moreover, as demonstrated earlier, the computation of λ is done in time $O(m^2 + n)$, and ELFJ runs in time $O(n \log n + mn)$, which gives a total time complexity of $O(m^2 + n \log n + mn)$. ■

In this proof, we avoid the error from Lin et al. by making sure that $b_j \leq \beta$ for all jobs j assigned between either a_{j_0} and β, or between a_{j_1} and β, or between a_{j_2} and β, etc. In the previous counterexample, we would have stopped at $\alpha = a_j$ and $\beta = b_j$.

4.3 An Approximation for Arbitrary Jobs

ELFJ is optimal for unitary jobs. It may well be applied to non-unitary jobs, but does not produce an optimal schedule. We prove here that it is however an approximation algorithm, with a small adaptation, for arbitrary processing times. In the following, we note $p_{\max}$ the maximum processing time among all jobs.

Theorem 2. *Let* $\lambda = \tilde{w}_{\max} + \left(1 - \frac{1}{m}\right) p_{\max}$. *Then ELFJ (Algorithm 1) is a* $(2 - 1/m)$*-approximation algorithm and runs in time* $O(m^2 + n \log n + mn)$.

Proof: Suppose by contradiction that ELFJ does not give a feasible schedule with makespan at most λ. Let j_0 be the first non-assigned job. Then all machines in $\mathcal{M}_{j_0}$ must finish after time $\lambda - p_{j_0}$, otherwise we would have assigned j_0. Let $\beta = b_{j_0}$, and let $\gamma \leq a_{j_0}$ be the smallest machine index such that all machines between γ and β complete after time $\lambda - p_{\max}$. This means that the machine $\gamma - 1$ completes at or before time $\lambda - p_{\max}$ if $\gamma > 1$. Hence, we have $a_j \geq \gamma$ for all jobs j assigned on a machine whose index is greater than γ, otherwise j would have been assigned on $\gamma - 1$ by ELFJ, as $p_j \leq p_{\max}$ (by definition of $p_{\max}$).

Now let $S(a, b, t)$ be the set of jobs assigned by ELFJ between machines a and b, and scheduled to start at or before time t ($S(a, b, t) = \emptyset$ if $a > b$). We can see this set $S(a, b, t)$ as a work area, whose minimal shape is the rectangle delimited by a, b and t. Our goal is to prove that there exists a machine α between γ and a_{j_0} such that all jobs in the set $S(\alpha, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$, whose minimal work area can be represented by two adjacent rectangles, come from the set $K_{(\alpha,\beta)}$, which includes all jobs whose processing set is in the interval (α, β). The Figure 2 highlights the work areas of interest.

To do so, we adapt the constructive process that we used in the previous proof. Let us prove that there exists a non-empty set of machines $u_1 > u_2 > \cdots > u_x$ between a_{j_0} and γ such that

- for all u_k, $b_j \leq \beta$ for all jobs $j \in S(u_k, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$, and
- for all $u_{k<x}$, there exists $j \in S(u_k, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$ such that $\gamma \leq a_j < u_k$, and
- $a_j \geq u_x$ for all $j \in S(u_x, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$.

Base case ($u_1 = a_{j_0}$). Let $j \in S(a_{j_0}, \beta, \lambda - p_{j_0})$ be a job assigned between a_{j_0} and β, and starting at or before $\lambda - p_{j_0}$. We have $b_j \leq b_{j_0} = \beta$, otherwise the job j_0 could have been scheduled instead of job j. Then, either $a_j \geq a_{j_0}$ for all $j \in S(a_{j_0}, \beta, \lambda - p_{j_0})$, or there is $j \in S(a_{j_0}, \beta, \lambda - p_{j_0})$ such that $\gamma \leq a_j < a_{j_0}$. In the first case, we set $x = 1$ and we are done. In the second case, we proceed to the next step.

Induction step. Suppose that $b_j \leq \beta$ for all $j \in S(u_k, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$. Moreover, suppose there exists $j_1 \in S(u_k, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$ such that $\gamma \leq a_{j_1} < u_k$ at step k. Let us choose j_1 such that a_{j_1} is minimal, and let $u_{k+1} = a_{j_1}$.

Now let $j_2 \in S(u_{k+1}, u_k - 1, \lambda - p_{\max})$ be any job assigned between machines u_{k+1} and $u_k - 1$. We have $b_{j_2} \leq b_{j_1}$, otherwise the job j_1 would have been scheduled instead of job j_2. Hence, $b_{j_2} \leq \beta$ (by induction hypothesis), thus $b_j \leq \beta$ for all $j \in S(u_{k+1}, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$, because $S(u_{k+1}, u_k - 1, \lambda - p_{\max}) \cup S(u_k, a_{j_0} - 1, \lambda - p_{\max}) = S(u_{k+1}, a_{j_0} - 1, \lambda - p_{\max})$.

Then, either $a_{j_2} \geq u_{k+1}$ for all $j_2 \in S(u_{k+1}, u_k - 1, \lambda - p_{\max})$, or there exists $j_2 \in S(u_{k+1}, u_k - 1, \lambda - p_{\max})$ such that $\gamma \leq a_{j_2} < u_{k+1}$. In the first case, we conclude that $a_j \geq u_{k+1}$ for all $j \in S(u_{k+1}, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$, because we have chosen j_1 in a way that a_{j_1} is minimal (thus $a_j \geq a_{j_1} = u_{k+1}$ for all $j \in S(u_k, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$). Hence, we set $x = k + 1$ and we stop there. In the second case, we proceed to the next step.

Therefore, we proved that we can find a machine u_x such that $a_j \geq u_x$ and $b_j \leq \beta$ for all $j \in S(u_x, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$. In other words, all jobs in the set $S(u_x, a_{j_0} - 1, \lambda - p_{\max}) \cup S(a_{j_0}, \beta, \lambda - p_{j_0})$ come from the set $K_{(u_x,\beta)}$.

Recall that all machines $a_{j_0}, a_{j_0} + 1, \ldots, \beta$ finish after time $\lambda - p_{j_0}$, and by construction, all machines $u_x, u_x + 1, \ldots, a_{j_0} - 1$ finish after time $\lambda - p_{\max}$, because $u_x \geq \gamma$. Thus we set $\alpha = u_x$, and we have

$$w_{(\alpha,\beta)} > (\beta - \alpha + 1)(\lambda - p_{\max}) + (\beta - a_{j_0} + 1)(\lambda - p_{j_0} - (\lambda - p_{\max})) + p_{j_0},$$

which gives the following inequality:

$$\lambda < \tilde{w}_{(\alpha,\beta)} - \frac{p_{j_0}}{\beta - \alpha + 1} - \frac{(\beta - a_{j_0} + 1)(p_{\max} - p_{j_0})}{\beta - \alpha + 1} + p_{\max}.$$

As $\beta - a_{j_0} + 1 \geq 1$ and $\beta - \alpha + 1 \leq m$, we have $\frac{\beta - a_{j_0} + 1}{\beta - \alpha + 1} \geq \frac{\beta - a_{j_0} + 1}{m} \geq \frac{1}{m}$. Moreover, $p_{\max} - p_{j_0} \geq 0$, then

$$\lambda < \tilde{w}_{(\alpha,\beta)} - \frac{p_{j_0}}{\beta - \alpha + 1} - \frac{1}{m}(p_{\max} - p_{j_0}) + p_{\max},$$

thus,

$$\lambda < \tilde{w}_{(\alpha,\beta)} + \left(1 - \frac{1}{m}\right) p_{\max} + \left(\frac{1}{m} - \frac{1}{\beta - \alpha + 1}\right) p_{j_0}.$$

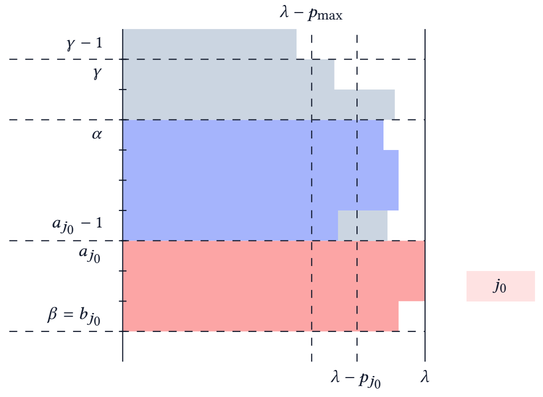

Figure 2: Work areas between machines α and β. The blue area is $S(\alpha, a_{j_0} - 1, \lambda - p_{\max})$. The red area is $S(a_{j_0}, \beta, \lambda - p_{j_0})$. Gray areas are the other jobs. We seek to prove that the blue and red areas are made of jobs included in $K_{(\alpha,\beta)}$.

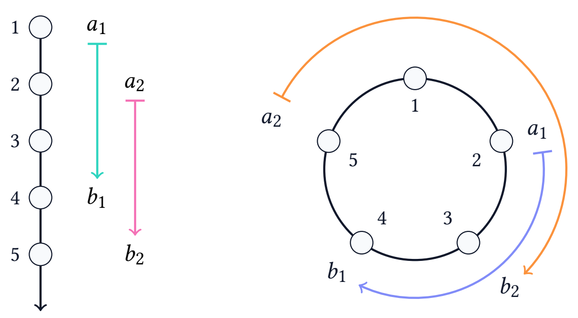

Figure 3: Comparison between instances of the standard RAI problem (on the left) and its generalization to circular intervals (on the right).

Finally, we have $\frac{1}{\beta-\alpha+1} \geq \frac{1}{m}$, i.e., $\frac{1}{m} - \frac{1}{\beta-\alpha+1} \leq 0$, and $p_{j_0} \geq 0$. Therefore,

$$\lambda < \tilde{w}_{(\alpha,\beta)} + \left(1 - \frac{1}{m}\right) p_{\max},$$

which is a contradiction.

Hence, ELFJ gives a schedule that is feasible in time λ, i.e., $C_{\max} \leq \lambda$. By Lemma 1, we have $C_{\max}^{OPT} \geq \tilde{w}_{\max}$, and obviously, $C_{\max}^{OPT} \geq p_{\max}$, so $\lambda \leq (2-1/m) \cdot C_{\max}^{OPT}$. We conclude that $C_{\max} \leq (2-1/m) \cdot C_{\max}^{OPT}$.

Note that this approximation ratio is tight. One may easily consider an instance with one job of size 2 and $2(m-1)$ unitary jobs (all feasible on all machines), i.e., $\lambda = 2(2-1/m)$. ELFJ will keep scheduling unitary jobs on the first machine until it reaches makespan $2(2-1/m)$, whereas the optimal makespan is 2. ∎

5 A GENERAL FRAMEWORK FOR CIRCULAR INTERVALS

In the standard RAI problem, machines are linearly arranged, that is to say, they are numbered from 1 to m and virtually placed on a line. As we have seen in the introduction, databases often organize machines in a virtual ring, where the machines able to answer a query for a particular key are arranged as a consecutive subset of this ring. We generalize here the notion of interval to take into account this setting by introducing the notion of *circular* intervals. Machines are now virtually arranged on a circle. In addition to regular intervals (a, b) (with $a \leq b$), we introduce *circular* intervals such that $a > b$. In this case, the corresponding set $I_{(a,b)}$ includes machines $a, a+1, \dots, m$ and machines $1, 2, \dots, b$, i.e., we have

$$I_{(a,b)} = \begin{cases} \{a, a+1, \dots, b\} & \text{if } a \leq b, \\ \{1, 2, \dots, b\} \cup \{a, a+1, \dots, m\} & \text{otherwise.} \end{cases}$$

Note that we clearly cannot always rearrange machines to transform an instance with circular intervals to an instance without circular intervals. Consider the instance with 3 machines and 3 jobs with processing sets $\mathcal{M}_1 = \{1, 2\}$, $\mathcal{M}_2 = \{2, 3\}$ and $\mathcal{M}_3 = \{3, 1\}$: any permutation of the machines will exhibit exactly one circular interval. Figure 3 illustrates the generalization of the RAI problem to circular intervals. By extension, we call this generalized problem the Restricted Assignment problem on Circular Intervals (RACI).

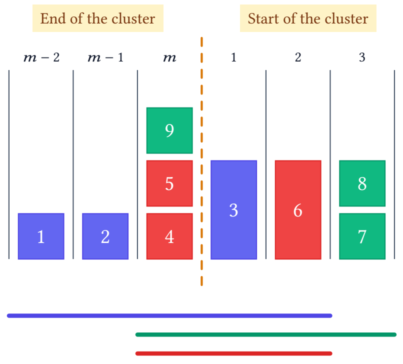

Figure 4: Example of circular jobs in a schedule. Colors denote jobs with common processing sets. Jobs 1, 2, 4, 5, 9 are left jobs, whereas jobs 3, 6, 7, 8 are right jobs. Moreover, there are two types of jobs. Jobs 1, 2, 4, 5, 7, 8, 9 are of type 1 (with $p(1) = 1$), whereas jobs 3 and 6 are of type 2 (with $p(2) = 2$). Thus, in this example, $G_1 = \{1, 2, 4, 5, 9\}$, $D_1 = \{7, 8\}$, $G_2 = \emptyset$, and $D_2 = \{3, 6\}$.

Definition 1. *The interval* (a_g, b_g) precedes *the interval* (a_h, b_h) *if and only if* $a_g \leq a_h$ *and* $b_g \leq b_h$. *In this case, we note* $(a_g, b_g) \preceq (a_h, b_h)$.

For a given instance, let Z^* be the set of circular intervals that are associated to at least one job ($Z^* = \{(a_j, b_j) \text{ s.t. } j \in J \text{ and } a_j > b_j\}$). In this section, we restrict ourselves to instances where the previously-defined relation $\preceq$ is a total order on Z^*. In other words, for any $g, h \in Z^*$, we cannot have $I_g \subset I_h$ or $I_h \subset I_g$. This constitutes a particular case of RACI, but it is still a more general case than RAI. Moreover, we assume that there are K types of jobs, and each job of type k has processing time $p(k)$. We introduce in this section a general procedure that solves the RACI problem for these restricted instances, assuming that one already knows an optimal algorithm $\mathcal{A}$ for the standard RAI problem with K job types.

Theorem 3. *Let $\mathcal{A}$ be an optimal algorithm for the RAI problem with K job types that runs in time $O(f(n))$, where f is a polynomial. Then there exists a procedure that solves the corresponding RACI problem on ordered circular intervals in time $O(n^K f(n))$.*

We begin with a few definitions. Then we present the procedure, before proving our result.

Preliminaries. Let J^* be the subset of jobs whose processing set is a circular interval, i.e., $J^* = \{j \in J \text{ s.t. } a_j > b_j\}$, and we note $n^* = |J^*|$. We call J^* the *circular* jobs. We also partition J^* into K subsets $J_1^*, \dots, J_K^*$, such that all jobs in J_k^* are of type k, and we note $n_k^* = \left|J_k^*\right|$.

Moreover, in a given schedule, we say that a circular job j assigned between a_j (inclusive) and m (inclusive) is a *left* job. Equivalently, a circular job j assigned between 1 (inclusive) and b_j (inclusive) is a *right* job. This means that a schedule S implicitely defines a partition of each set J_k^* into two subsets G_k and D_k, where G_k contains γ_k left jobs, and D_k contains δ_k right jobs. Figure 4 shows an example of such schedule.

We present here the intuition on how to compute an optimal schedule by considering all possible partitions of jobs with circular processing sets into left and right jobs. For the moment, we simplify the problem by considering only one type of jobs. A schedule defines a partition of jobs J^* into left and right jobs, which means that we need to find *how many* jobs in J^* should be assigned to the left or to the right. Thus, assume that we know that r jobs of J^* must be assigned to the right in an optimal schedule. Intuitively, the r circular jobs with rightmost intervals should be put on the right, and the remaining jobs of J^* should be put on the left. For example, consider only the small jobs in the instance of Figure 4. If we suppose that $r = 5$ (arbitrarily), then we guess that the 2 red jobs and the 3 green jobs should be put on the right (i.e., between machines 1 and 3), and the 2 blue jobs should be put on the left (i.e., between machines $m-2$ and m), as the red and green intervals are more on the right than the blue interval. We introduce below the notion of *right-sorted* schedules that captures this intuition, and we will prove later that there always exists optimal schedules that have this property.

Definition 2. *A schedule S is* right-sorted *if and only if for each type k, the property $(a_j, b_j) \preceq (a_{j'}, b_{j'})$ holds for any jobs $j \in G_k$ and $j' \in D_k$.*

We denote the set of all possible schedules for a given instance $\mathcal{I}$ by $\mathcal{S}(\mathcal{I})$ ($\mathcal{I}$ is omitted when it is clear from the context). Let $\mathbf{R}^K$ be the set of all vectors $\mathbf{r} = (r_1, \cdots, r_K)$ such that r_k is an integer and $0 \le r_k \le n_k^*$ for all k. For a given vector $\mathbf{r} \in \mathbf{R}^K$, we call $\mathcal{S}_\mathbf{r}$ the subset of schedules $\mathcal{S}$ that put exactly r_k jobs of type k on the right, and $n_k^* - r_k$ jobs of type k on the left. Recall that $C_{\max}^{OPT}$ denotes the optimal makespan among all schedules $\mathcal{S}$. We define analogously $C_\mathbf{r}^{BEST}$ as the *best possible makespan* among schedules $\mathcal{S}_\mathbf{r}$. Note that the subsets $\mathcal{S}_\mathbf{r}$ define a partition of $\mathcal{S}$, and thus

$$C_{\max}^{OPT} = \min_{\mathbf{r} \in \mathbf{R}^K} \left\{ C_\mathbf{r}^{BEST} \right\}. \quad (1)$$

Optimal procedure. We introduce a polynomial procedure $\phi_\mathbf{r}$ that transforms any instance $\mathcal{I}$ of the (ordered) RACI problem into another instance $\mathcal{I}' = \phi_\mathbf{r}(\mathcal{I})$ that does not include any circular interval. We prove later the following two properties on the obtained instance:

(i) Applying an optimal algorithm $\mathcal{A}$ to $\mathcal{I}'$ produces a valid solution for $\mathcal{I}$.
(ii) The makespan of this solution is at most $C_\mathbf{r}^{BEST}$.

Given these properties and Equation (1), we can find an optimal solution for $\mathcal{I}$ by performing an exhaustive search of the best vector $\mathbf{r} \in \mathbf{R}^K$. For a given instance $\mathcal{I}$, the function $\phi_\mathbf{r}$ works as follows:

(1) Sort jobs J^* by non-increasing order of b_j, and sort jobs with identical b_j by non-increasing order of a_j. Note that this corresponds to sorting jobs by non-increasing order of $\preceq$. As $\preceq$ is a total order on Z^*, all jobs are comparable.
(2) For each type k, set $a_j = 1$ for the r_k first jobs of J_k^*, and $b_j = m$ for the $n_k^* - r_k$ other jobs.

Let $\mathcal{S}_\mathbf{r}^{\preceq}$ be the subset of schedules $\mathcal{S}_\mathbf{r}$ that are right-sorted. The proof of the two properties of interest is structured as follows. As $\mathcal{A}$ is optimal for the standard RAI problem, we know that it finds one of the best schedules among $\mathcal{S}(\phi_\mathbf{r}(\mathcal{I}))$. Hence, we will prove two lemmas. On the one hand, we show in Lemma 2 that the set $\mathcal{S}(\phi_\mathbf{r}(\mathcal{I}))$ is exactly the same as the set of right-sorted schedules $\mathcal{S}_\mathbf{r}^{\preceq}(\mathcal{I})$ for the initial instance. On the other hand, we show in Lemma 3 that there always exists a right-sorted schedule that has the best possible makespan.

Lemma 2. *For any $\mathbf{r} \in \mathbf{R}^K$, we have $\mathcal{S}(\phi_\mathbf{r}(\mathcal{I})) = \mathcal{S}_\mathbf{r}^{\preceq}(\mathcal{I})$.*

Proof: Let $\mathbf{r}$ be an arbitrary vector of $\mathbf{R}^K$. First we show that $\mathcal{S}(\phi_\mathbf{r}(\mathcal{I})) \subseteq \mathcal{S}_\mathbf{r}^{\preceq}(\mathcal{I})$. Let $S \in \mathcal{S}(\phi_\mathbf{r}(\mathcal{I}))$. By definition of $\phi_\mathbf{r}$, for all types k, there are $n_k^* - r_k$ jobs in S that were circular jobs in the initial instance $\mathcal{I}$ and that are on the left (similarly, there are r_k jobs in S that were circular and that are on the right). Moreover, the circular jobs have been sorted in $\phi_\mathbf{r}$, which means that for all k, we have $(a_j, b_j) \preceq (a_{j'}, b_{j'})$ for any $j \in G_k$ and $j' \in D_k$ in S. In other words, S is right-sorted, and thus belongs to $\mathcal{S}_\mathbf{r}^{\preceq}$.

Now we show that $\mathcal{S}_\mathbf{r}^{\preceq}(\mathcal{I}) \subseteq \mathcal{S}(\phi_\mathbf{r}(\mathcal{I}))$. By definition of $\mathcal{S}_\mathbf{r}^{\preceq}$, in any schedule $S \in \mathcal{S}_\mathbf{r}^{\preceq}(\mathcal{I})$, for all types k, we have $(a_j, b_j) \preceq (a_{j'}, b_{j'})$ for any $j \in G_k$ and $j' \in D_k$. Moreover, there are exactly $n_k^* - r_k$ jobs in G_k and r_k jobs in D_k. Thus, S is clearly a valid solution for $\phi_\mathbf{r}(\mathcal{I})$ and belongs to $\mathcal{S}(\phi_\mathbf{r}(\mathcal{I}))$. ∎

Lemma 3. *For any $\mathbf{r} \in \mathbf{R}^K$, there exists a right-sorted schedule $S \in \mathcal{S}_\mathbf{r}^{\preceq}$ that has the best possible makespan $C_\mathbf{r}^{BEST}$.*

Proof: Let $\mathbf{r}$ be an arbitrary vector of $\mathbf{R}^K$. Let $S \in \mathcal{S}_\mathbf{r}$ be a schedule that has the best possible makespan $C_\mathbf{r}^{BEST}$. If S is right-sorted, we are done. Otherwise, there necessarily exists a type k such that two jobs $j \in G_k$ and $j' \in D_k$, scheduled in S, are not sorted according to $\preceq$, i.e., we have $(a_j, b_j) \npreceq (a_{j'}, b_{j'})$. In other words, either $a_j > a_{j'}$, or $b_j > b_{j'}$. We know that $\preceq$ is a total order on Z^*, which means that if $a_j > a_{j'}$, then we necessarily have $b_j \ge b_{j'}$. In a similar way, if $b_j > b_{j'}$, then we necessarily have $a_j \ge a_{j'}$. This means that even if j is a left job and j' is a right job in S, there is "more room" to put j on the right side and j' on the left side of their respective interval. Moreover, as j and j' have the same type k, they have identical processing times. Hence, we can clearly swap j and j' in S without changing the makespan of S.

By repeatedly swapping non-sorted jobs of the same type, we reach another schedule S' that has the same makespan than S, that also belongs to $\mathcal{S}_\mathbf{r}$, and that is right-sorted. ∎

Now we are able to conclude. By hypothesis, we know that $\mathcal{A}$ finds a schedule with the smallest makespan among $\mathcal{S}(\phi_\mathbf{r}(\mathcal{I}))$. By Lemma 3, we also know that there exists at least one schedule in $\mathcal{S}_\mathbf{r}^{\preceq}(\mathcal{I})$ that has the best possible makespan $C_\mathbf{r}^{BEST}$. Therefore, we deduce by Lemma 2 that:

- the solution given by $\mathcal{A}$, when applied to $\phi_\mathbf{r}(\mathcal{I})$, belongs to $\mathcal{S}_\mathbf{r}^{\preceq}(\mathcal{I})$, which means that it also belongs to $\mathcal{S}_\mathbf{r}(\mathcal{I})$, i.e., it is a valid solution for $\mathcal{I}$ (Property (i)), and
- the solution given by $\mathcal{A}$, when applied to $\phi_\mathbf{r}(\mathcal{I})$, has makespan $C_\mathbf{r}^{BEST}$ (Property (ii)).

It follows that, for any instance $\mathcal{I}$, we can find the best possible schedule among $\mathcal{S}_\mathbf{r}(\mathcal{I})$ for any vector $\mathbf{r} \in \mathbf{R}^K$ in polynomial time, as the transformation procedure $\phi_\mathbf{r}$ and the algorithm $\mathcal{A}$ are themselves polynomial.

Moreover, for all vectors $\mathbf{r} \in \mathbf{R}^K$, we have $r_k \le n_k^* \le n$ for all k. Thus, the number of possible vectors $\mathbf{r}$ is bounded by $O(n^K)$, i.e., we can find an optimal schedule for any instance $\mathcal{I}$ by searching over all possible vectors in time $O(n^K f(n))$, assuming that we know a polynomial algorithm $\mathcal{A}$ that runs in time $O(f(n))$ when applied to $\phi_{\mathbf{r}}(\mathcal{I})$. This concludes the proof of Theorem 3.

We now study two special cases where this procedure can be applied: the adaptation of an existing dynamic programming algorithm for K job types and the ELFJ algorithm presented above. In the latter case, we are able to largely reduce the complexity compared to Theorem 3, as we achieve for ELFJ on circular intervals the same complexity as ELFJ on regular intervals.

5.1 Adapting an Optimal Dynamic Program

We illustrate how our framework can be successfully applied to derive a polynomial algorithm for the RACI problem on intervals of equal length and K job types. Wang et al. [14] showed how to solve the corresponding problem on non-circular intervals with a dynamic program. For completeness, we recall their solution in the following.

Let n_k be the number of jobs of type k ($1 \leq k \leq K$), and let us sort jobs by non-decreasing value of b_j. Suppose that λ is a value that represents a hard deadline for all jobs. Define $F_i(s_1, s_2, \ldots, s_K) = 1$ if and only if it is feasible, for all types k, to schedule s_k jobs of type k on machines $1, 2, \ldots, i$ such that the makespan is at most λ, and $F_i(s_1, s_2, \ldots, s_K) = 0$ otherwise.

Let $F_0(0, \ldots, 0) = 1$, and $F_i(s_1, s_2, \ldots, s_K) = 1$ if and only if there exist $s'_1 \leq s_1, s'_2 \leq s_2, \ldots, s'_K \leq s_K$ such that:

(i) $F_{i-1}(s'_1, s'_2, \ldots, s'_K) = 1$,
(ii) for each k, the next $s_k - s'_k$ jobs of type k in the sorted set contain the machine i in their processing set, and
(iii) $\sum_{k=1}^{K} (s_k - s'_k) \cdot p(k) \leq \lambda$.

Then we have $F_m(n_1, n_2, \ldots, n_K) = 1$ if and only if there exists a schedule feasible in time λ. For a given value of λ, an array with all values of F can be computed in time $O(mn^{2K})$. Finally, the optimal value of λ can be found by performing a binary search. Thus the overall complexity of the algorithm is $O(mn^{2K} \log \sum p_j)$.

By using our framework, adapting this approach to the RACI problem is straightforward. Let $\mathcal{A}$ be the dynamic program described above. By Theorem 3, we know that we can find an optimal schedule for any instance in time $O(n^K f(n))$, where $f(n)$ is the complexity of $\mathcal{A}$. Therefore, the time complexity of the derived algorithm is $O(mn^{3K} \log \sum p_j)$.

5.2 Revisiting ELFJ

We proved in Theorem 1 that ELFJ is an optimal algorithm for the standard RAI problem on unitary jobs, which runs in time $f(n) = O(m^2 + n \log n + mn)$. Recall that ELFJ consists in 3 distinct steps:

(1) computing the optimal makespan value, in time $O(m^2 + n)$,
(2) sorting the jobs, in time $O(n \log n)$,
(3) performing the actual job assignment, in time $O(mn)$.

By applying our framework around ELFJ, and because we have only one type of jobs in this specific case, we know from Theorem 3 that we can solve the generalized problem on ordered circular intervals in time $O(nf(n)) = O(m^2 n + n^2 \log n + mn^2)$. We now show how to

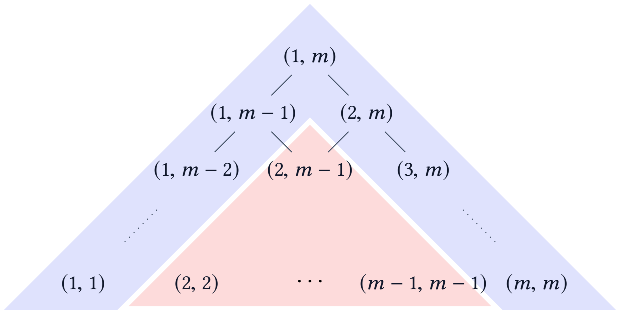

Figure 5: Outside and inside intervals in the lattice graph representation. Outside intervals (blue area) are of the form $(1, x)$ or (x, m), with $1 \leq x \leq m$, and inside intervals (red area) are of the form (x, y), with $1 < x \leq y < m$.

improve the complexity of this solution, as stated in the following theorem.

Theorem 4. *The ordered RACI problem with unitary jobs can be solved in time $O(m^2 + n \log n + mn)$.*

Proof: The basic idea is to extract and reorganize some internal computation steps from the exhaustive search procedure to avoid doing any redundant work. We first observe that the only step of ELFJ that actually depends on knowing an optimal number r of right jobs (in the set of circular jobs) is the computation of the optimal makespan. Once we know the best values of r and λ, the sorting and job assignment steps are straightforward. Thus we know that we can easily refine the complexity to $O(n(m^2+n)+n\log n+mn) = O(m^2 n + n^2)$.

To reduce further the complexity, we notice that we do not really need to recompute the matrix w from the beginning (in time $O(m^2+n)$) for each possible value of r in order to find the minimum makespan. We remark that there are two kinds of non-circular intervals: the ones that may result from cutting a circular interval (a, b) in two subintervals (a, m) and $(1, b)$, which we call the *outside* intervals, and the ones that cannot, which we call *inside* intervals. In other words, outside intervals are all non-circular intervals of the form $(1, x)$ or (x, m), with $1 \leq x \leq m$, and inside intervals are all non-circular intervals of the form (x, y) with $1 < x \leq y < m$. When representing the non-circular interval hierarchy as a lattice graph, the outside intervals are in fact all the nodes on the sides of the lattice, and the inside intervals are the others, as shown in Figure 5.

Recall that $w_{(\alpha,\beta)}$ represents the total work of all non-circular jobs whose interval is included in (α, β). When we update our guess on the optimal number of right jobs, we transform the instance by shrinking the intervals of circular jobs: if a job j is a right job, we keep the right part of the interval, i.e., the subinterval $(1, b_j)$, and if it is a left job, we keep the left part of the interval, i.e., the subinterval (a_j, m). This means that the only values of w that may change when we transform the instance are the ones that are associated to the outside intervals. All other values remain unchanged, no matter how we partition the circular jobs.

Hence, we can decompose the computation of $\tilde{w}_{\max}$ in two steps. First, compute the value

$$\tilde{w}_{\max}^{inside} = \max_{1<\alpha\leq\beta<m} \left\{ \tilde{w}_{(\alpha,\beta)} \right\},$$

which represents the maximum value of $\tilde{w}$ among all inside intervals. This value does not depend on r, and can be computed only once. Second, compute the value

$$\tilde{w}_{\max}^{outside} = \max_{1\leq x\leq m} \left\{ \max\left(\tilde{w}_{(1,x)}, \tilde{w}_{(x,m)} \right) \right\},$$

which represents the maximum value of $\tilde{w}$ among all outside intervals. We clearly have

$$\tilde{w}_{\max} = \max\left(\tilde{w}_{\max}^{inside}, \tilde{w}_{\max}^{outside} \right).$$

In other words, each time we update our guess on r, we only need to recompute the value of $\tilde{w}_{\max}^{outside}$, which can be done in time $O(m)$ as there are exactly $2m-1$ outside intervals. Thus, we can search for the minimum value of $\max\left(\tilde{w}_{\max}^{inside}, \tilde{w}_{\max}^{outside} \right)$ by pre-computing $\tilde{w}_{\max}^{inside}$, and then trying each possible value of r by updating only $\tilde{w}_{\max}^{outside}$.

The only remaining question is how we know which values to update in the matrix w when we make a new guess on r. We avoid recomputing the values that are associated to inside intervals. We can also avoid recomputing the value $w_{(1,m)}$, as it is always exactly equal to n, and we have $\tilde{w}_{(1,m)} = n/m$. Recall that the set of circular jobs is sorted by decreasing order of $\preceq$ (by definition of $\phi_{\mathbf{r}}$), and for a given number r, we know that the first r jobs in the sorted set are right jobs. We set $r = 0$ and we pre-compute w, $\tilde{w}_{\max}^{inside}$ and $\tilde{w}_{\max}$. Then we loop over the sorted set of circular jobs by adding them progressively on the right side, i.e., for each job $j \in J^*$, we add 1 to $w_{(1,\beta)}$ for all $b_j \leq \beta < m$ and we subtract 1 to $w_{(\alpha,m)}$ for all $1 < \alpha \leq a_j$. The full procedure is given in Algorithm 3.

We conclude that the RACI problem with ordered circular intervals and unitary jobs can be solved in time $O(m^2 + n\log n + mn)$, or $O(n \log n)$ if we assume that m is fixed. ■

Algorithm 3 Computing r and $\tilde{w}_{\max}$ in time $O(m^2 + n\log n + mn)$

```
1:  sort circular jobs by decreasing order of $\preceq$
2:  transform circular jobs as left jobs and pre-compute $w$, $\tilde{w}_{\max}^{inside}$
    and $\tilde{w}_{\max}$
3:  $r_{cur} \leftarrow 0$
4:  for each circular job $j \in J^*$ do
5:      $\tilde{w}_{\max}^{outside} \leftarrow \tilde{w}_{(1,m)}$
6:      for each $\beta$ from $b_j$ to $m-1$ do            ▷ Add $j$ on the right
7:          $w_{(1,\beta)} \leftarrow w_{(1,\beta)} + 1$
8:          $\tilde{w}_{(1,\beta)} \leftarrow \frac{w_{(1,\beta)}}{\beta}$
9:          if $\tilde{w}_{(1,\beta)} > \tilde{w}_{\max}^{outside}$ then
10:             $\tilde{w}_{\max}^{outside} \leftarrow \tilde{w}_{(1,\beta)}$
11:     for each $\alpha$ from 2 to $a_j$ do            ▷ Remove $j$ from the left
12:         $w_{(\alpha,m)} \leftarrow w_{(\alpha,m)} - 1$
13:         $\tilde{w}_{(\alpha,m)} \leftarrow \frac{w_{(\alpha,m)}}{m-\alpha+1}$
14:         if $\tilde{w}_{(\alpha,m)} > \tilde{w}_{\max}^{outside}$ then
15:             $\tilde{w}_{\max}^{outside} \leftarrow \tilde{w}_{(\alpha,m)}$
16:     $r_{cur} \leftarrow r_{cur} + 1$
17:     $\tilde{w}_{cur} \leftarrow \max\left(\tilde{w}_{\max}^{inside}, \tilde{w}_{\max}^{outside}\right)$
18:     if $\tilde{w}_{cur} < \tilde{w}_{\max}$ then            ▷ Update minimum makespan
19:         $r \leftarrow r_{cur}$
20:         $\tilde{w}_{\max} \leftarrow \tilde{w}_{cur}$
```

6 CONCLUSION

In this paper, we improved prior work done on the Restricted Assignment problem on Intervals by giving a generalized version of an existing algorithm. Our version solves the problem with unitary jobs to optimality in polynomial time $O(m^2 + n\log n + mn)$, and we proved that it also provides a $(2 - 1/m)$-approximation in the general case. Moreover, we extended the RAI problem to *circular* intervals that corresponds to common use cases in distributed databases and is thus of particular practical interest. We proposed a general framework that, given an optimal algorithm for the RAI problem with at most K job types and running in time $O(f(n))$, computes an optimal solution for the RAI problem with circular intervals (RACI) in time $n^K f(n)$. This enabled us to revisit the initial algorithm for the RAI problem with unitary jobs to derive an optimal algorithm for the RACI problem with unitary jobs, also running in time $O(m^2 + n\log n + mn)$.

We identified several challenges for further explorations. First, it is unknown if there exists a reasonably efficient algorithm for the RAI problem with two types of jobs, which would be very suitable for multi-get request partitioning in key-value store systems, as workloads often exhibit long-tailed distributions, with lots of small jobs and some that are a lot bigger. Second, it would be interesting to know if there is an efficient (i.e., usable in practice) approximation algorithm that improves on the $2 - 1/m$ guaranteed factor. Finally, we identified that instances of the RACI problem where some circular intervals are strictly included into other circular intervals are a lot more challenging to solve than the ordered case. The unitary jobs variant is polynomial, but it is still unknown if this is the case when considering at least two job types.

REFERENCES

[1] Péter Biró and Eric McDermid. 2014. Matching with sizes (or scheduling with processing set restrictions). *Discrete Applied Mathematics* 164 (2014), 61–67.

[2] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In *SOSP 2007*. 205–220.

[3] Tomás Ebenlendr, Marek Krcál, and Jiri Sgall. 2008. Graph balancing: a special case of scheduling unrelated parallel machines.. In *SODA*, Vol. 8. 483–490.

[4] Leah Epstein and Asaf Levin. 2011. Scheduling with processing set restrictions: PTAS results for several variants. *International Journal of Production Economics* 133, 2 (2011), 586–595.

[5] Celia A Glass and Hans Kellerer. 2007. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics* 54, 3 (2007), 250–257.

[6] Vikas Jaiman, Sonia Ben Mokhtar, and Etienne Rivière. 2020. TailX: Scheduling heterogeneous multiget queries to improve tail latencies in key-value stores. In *IFIP International Conference on Distributed Applications and Interoperable Systems*. 73–92.

[7] Klaus Jansen and Lars Rohwedder. 2017. Structured Instances of Restricted Assignment with Two Processing Times. In *Conference on Algorithms and Discrete Applied Mathematics*. 230–241.

[8] Klaus Jansen and Lars Rohwedder. 2020. A Quasi-Polynomial Approximation for the Restricted Assignment Problem. *SIAM J. Comput.* 49, 6 (2020), 1083–1108.

[9] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40.

[10] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. 1990. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical programming*

46, 1 (1990), 259–271.

[11] Yixun Lin and Wenhua Li. 2004. Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research* 156, 1 (2004), 261–266.

[12] Marten Maack and Klaus Jansen. 2019. Inapproximability results for scheduling with interval and resource restrictions. *arXiv preprint arXiv:1907.03526* (2019).

[13] Waleed Reda, Marco Canini, Lalith Suresh, Dejan Kostić, and Sean Braithwaite. 2017. Rein: Taming tail latency in key-value stores via multiget scheduling. In *Proceedings of the Twelfth European Conference on Computer Systems*. 95–110.

[14] Chao Wang and René Sitters. 2016. On some special cases of the restricted assignment problem. *Inform. Process. Lett.* 116, 11 (2016), 723–728.