Improved Algorithms for Scheduling Unsplittable Flows on Paths*†

Hamidreza Jahanjou[1], Erez Kantor[2], and Rajmohan Rajaraman[3]

1 **Northeastern University, Boston, USA**
hamid@ccs.neu.edu
2 **University of Massachusetts, Amherst, USA**
erez.kantor@gmail.com
3 **Northeastern University, Boston, USA**
rraj@ccs.neu.edu

Abstract

In this paper, we investigate offline and online algorithms for Round-UFPP, the problem of minimizing the number of rounds required to schedule a set of unsplittable flows of non-uniform sizes on a given path with non-uniform edge capacities. Round-UFPP is NP-hard and constant-factor approximation algorithms are known under the no bottleneck assumption (NBA), which stipulates that maximum size of a flow is at most the minimum edge capacity. We study Round-UFPP *without the NBA*, and present improved online and offline algorithms. We first study offline Round-UFPP for a restricted class of instances called α-small, where the size of each flow is at most α times the capacity of its bottleneck edge, and present an $O(\log(1/(1-\alpha)))$-approximation algorithm. Our main result is an online $O(\log\log c_{\max})$-competitive algorithm for Round-UFPP for general instances, where $c_{\max}$ is the largest edge capacities, improving upon the previous best bound of $O(\log c_{\max})$ due to [16]. Our result leads to an offline $O(\min(\log n, \log m, \log\log c_{\max}))$-approximation algorithm and an online $O(\min(\log m, \log\log c_{\max}))$-competitive algorithm for Round-UFPP, where n is the number of flows and m is the number of edges.

1998 ACM Subject Classification F.2.2 Nonnumerical algorithms and Problems

Keywords and phrases Approximation algorithms, Online algorithms, Unsplittable flows, Interval coloring, Flow scheduling

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.49

1 Introduction

The *unsplittable flow problem on paths* (UFPP) considers selecting a maximum-weight subset of flows to be routed simultaneously over a path while satisfying capacity constraints on the edges of the path. In this work, we investigate a variant of UFPP known in the literature as Round-UFPP or *capacitated interval coloring*. The objective in Round-UFPP is to schedule *all* the flows in the smallest number of rounds, subject to the constraint that the flows scheduled in a given round together respect edge capacities. Formally, in Round-UFPP we are given a path $P = (V, E)$, consisting of m links, with capacities $\{c_j\}_{j\in[m]}$, and a set of n flows $\mathcal{F} = \{f_i = (s_i, t_i, \sigma_i) : i \in [n]\}$ each consisting of a source vertex, a sink vertex, and a size. A set R of flows is feasible if all of its members can be scheduled simultaneously while satisfying

* This work was partially supported by NSF grant CCF-1422715, a Google Research Award, and an ONR grant on network algorithms.
† A full version of the paper is available at https://arxiv.org/abs/1708.00143.

capacity constraints. The objective is to partition $\mathcal{F}$ into the smallest number of feasible sets (rounds) $R_1, ..., R_t$.

One practical motivation for Round-UFP is routing in optical networks. Specifically, a flow f_i of size σ_i can be regarded as a connection request asking for a bandwidth of size σ_i. Connections using the same communication link can be routed at the same time as long as the total bandwidth requested is at most the link capacity. Most modern networks have heterogeneous link capacities; for example, some links might be older than others. In this setting, each round (or color) corresponds to a transmission frequency, and minimizing the number of frequencies is a natural objective in optical networks.

A common simplifying assumption, known as the no-bottleneck assumption (NBA), stipulates that the maximum demand size is at most the (global) minimum link capacity; i.e. $\max_{i\in[n]} \sigma_i \leq \min_{j\in[m]} c_j$; most results on UFPP and its variants are under the NBA (see §1.1). A major breakthrough was the design of $O(1)$-approximation algorithms for the unsplittable flow problem on paths (UFPP) without the NBA [10, 2]. In this paper, we make progress towards an optimal algorithm for Round-UFPP *without* imposing NBA.

We consider both offline and online versions of Round-UFPP. In the offline case, all flows are known in advance. In the online case, however, the flows are not known *à priori* and they appear one at a time. Moreover, every flow must be scheduled (i.e. assigned to a partition) immediately on arrival; no further changes to the schedule are allowed.

Even the simpler problem Round-UFPP-NBA, that is, Round-UFPP with the NBA, in the offline case, is **NP**-hard since it contains Bin Packing as a special case (consider an instance with a single edge). On the other hand, if all capacities and flow sizes are equal, then the problem reduces to interval coloring which is solvable by a simple greedy algorithm.

1.1 Previous work

The unsplittable flow problem on paths (UFPP) concerns selecting a maximum-weight subset of flows without violating edge capacities. UFPP is a special case of UFP, the unsplittable flow problem on general graphs. The term, *unsplittable* refers to the requirement that each flow must be routed on a single path from source to sink. [1] UFPP, especially under the NBA, UFPP-NBA, and its variants have been extensively studied [9, 3, 7, 6, 8, 11, 14, 22, 13]. Recently, $O(1)$-approximation algorithms were discovered for UFPP (without NBA) [10, 2]. Note that, on general graphs, UFP-NBA is **APX**-hard even on depth-3 trees where all demands are 1 and all edge capacities are either 1 or 2 [18].

Round-UFPP has been mostly studied in the online setting where it generalizes the interval coloring problem (ICP) which corresponds to the case where all demands and capacities are equal. In their seminal work, Kierstead and Trotter gave an optimal online algorithm for ICP with a competitive ratio of $3\omega - 2$, where ω denotes the maximum clique size [20]. Note that, since interval graphs are prefect, the optimal solution is simply ω. Many works consider the performance of the first-fit algorithm on interval graphs. Adamy and Erlebach were the first to generalize ICP [1]. In their problem, interval coloring with bandwidth, all capacities are 1 and each flow f_i has a size $\sigma_i \in (0, 1]$. The best competitive ratio known for this problem is 10 [5, 17] and a lower bound of slightly greater than 3 is known [19]. The online Round-UFPP is considered in Epstein et. al. [16]. They give a 78-competitive algorithm for Round-UFPP-NBA, an $O(\log \frac{\sigma_{\max}}{c_{\min}})$-competitive algorithm for the general Round-UFPP, and

[1] Clearly, in the case of paths and trees, the term is redundant. We use the terminology UFPP to be consistent with the considerable prior work in this area.

lower bounds of $\Omega(\log\log n)$ and $\Omega(\log\log\log \frac{c_{\max}}{c_{\min}})$ on the competitive ratio achievable for Round-UFPP. In the offline setting, a 24-approximation algorithm for Round-UFPP-NBA is presented in [15].

1.2 Our results

We design improved algorithms for offline and online Round-UFPP. Let m denote the number of edges in the path, n the number of flows, and $c_{\max}$ the maximum edge capacity.

- In §3, we design an $O(log(1/(1-\alpha)))$-approximation algorithm for offline Round-UFPP for α-small instances in which the size of each flow is at most an α fraction of the capacity of the smallest edge used by the flow, where $0 < \alpha < 1$. This implies an $O(1)$-approximation for any α-small instance, with constant α. Previously, constant-factor approximations were only known for $\alpha \leq 1/4$.
- In §4, we present our main result, an online $O(\log\log c_{\max})$-competitive algorithm for general instances. This result leads to an offline $O(\min(\log n, \log m, \log\log c_{\max}))$-approximation algorithm and an online $O(\min(\log m, \log\log c_{\max}))$-competitive algorithm.

Our algorithm for general instances, which improves on the $O(\log c_{\max})$-bound achieved in [16], is based on a reduction to the classic rectangle coloring problem (e.g., see [4, 21, 12]). We introduce a class of "line-sparse" instances of rectangle coloring that may be of independent interest, and show how competitive algorithms for such instances lead to competitive algorithms for Round-UFPP.

Due to space limitations, we are unable to include all of the proofs in the main body; we refer the reader to the full version of this paper[2] for any missing proof and pseudocode as well as extra figures.

2 Preliminaries

In Round-UFPP we are given a path $P = (V, E)$ consisting of $m + 1$ vertices and m links, enumerated left-to-right as $v_0, e_1, v_1, ..., v_{m-1}, e_m, v_m$, with edge capacities $\{c_j\}_{j\in[m]}$, and a set of n flows $\mathcal{F} = \{f_i = (s_i, t_i, \sigma_i) : i \in [n]\}$, where s_i and t_i represent the two endpoints of flow f_i, and σ_i denotes the size of the flow. Without loss of generality, we assume that $s_i < t_i$. We say that a flow f_i uses a link e_j if $s_i < j \leq t_i$. For a set of flows F, we denote by $F(e)$ and $F(j)$ the subset of flows in F using edge e and e_j respectively.

▶ **Definition 1.** The *bottleneck capacity* of a flow f_i, denoted by b_i, is the smallest capacity among all links used by f_i – such an edge is called the bottleneck edge for flow f_i.

A set of flows R is called *feasible* if all of its members can be routed simultaneously without causing capacity violation. The objective is to partition $\mathcal{F}$ into the smallest number of feasible sets $R_1, ..., R_t$. A feasible set is also referred to as a *round*.

Alternatively, partitioning can be seen as coloring where rounds correspond to colors.

▶ **Definition 2.** For a set of flows F, we define its *chromatic number*, $\chi(F)$, to be smallest number of rounds (colors) into which F can be partitioned.

▶ **Definition 3.** The *congestion* of an edge e_j with respect to a set of flows F is

$$r_j(F) = \frac{\sum_{i\in F(j)} \sigma_i}{c_j}, \tag{1}$$

[2] https://arxiv.org/abs/1708.00143

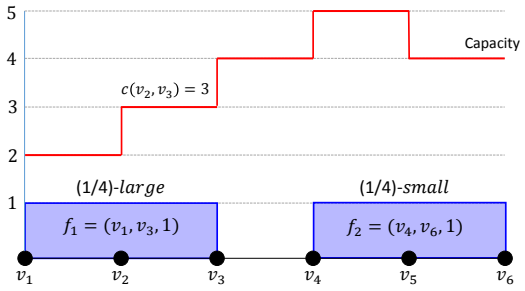

Figure 1 An example of a path with 5 links and two flows. The first flow f_1 is from v_1 to v_3 of size 1; the second flow f_2 is from v_4 to v_6 also of size 1. Even though both flows have the same size, f_1 is $\frac{1}{4}$-large whereas f_2 is $\frac{1}{4}$-small. The reason is different bottleneck capacities, $b_1 = 2$ and $b_2 = 4$.

that is, the ratio of the total size of flows in F using e_j to its capacity. Likewise $r_e(F)$ denotes the congestion of an edge e with respect to F. Also, let $r_{\max}(F) = \max_j r_j(F)$ be the maximum edge congestion with respect to F. When the set of flows is clear from the context, we simply write $r_{\max}$.

An obvious lower bound on $\chi(\mathcal{F})$ is maximum edge congestion; that is,

▶ **Observation 4.** $\chi(\mathcal{F}) \geq \lceil r_{\max}(\mathcal{F}) \rceil$.

Proof. Suppose e_j is any edge of the path. In each round, the amount of flow passing through the edge is at most its capacity c_j. Therefore, the number of rounds required for the flows in F using e_j to be scheduled is at least $\lceil r_j(F) \rceil$. ◀

Without loss of generality, we assume that the minimum capacity, $c_{\min}$, is 1. Furthermore, let $c_{\max} = \max_{e \in E} c_e$ denote the maximum edge capacity. As is standard in the literature, we classify flows according to the ratio of size to bottleneck capacity.

▶ **Definition 5.** Let α be a real number satisfying $0 \leq \alpha \leq 1$. A flow f_i is said to be *α-small* if $\sigma_i \leq \alpha \cdot b_i$ and *α-large* if $\sigma_i > \alpha \cdot b_i$ (refer to Figure 1 for an example). Accordingly, the set of flows $\mathcal{F}$ is divided into small and large classes

$$F_\alpha^S = \{f \in F \mid f \text{ is } \alpha\text{-small}\}; \quad F_\alpha^L = \{f \in F \mid f \text{ is } \alpha\text{-large}\}.$$

As is often the case for unsplittable flow algorithms, we treat small and large instances independently. In §3 and §4 we study small and large instances respectively.

3 An approximation algorithm for Round-UFPP with α-small flows

In this section, we design an offline $O(1)$-approximation algorithm for α-small flows for *any* $\alpha \in (0, 1)$. We note that offline and online algorithms for α-small instances are known when α is sufficiently small. More precisely, if $\alpha = 1/4$, 16-approximation and 32-competitive algorithms for offline and online cases have been presented in [15] and [16] respectively.

▶ **Lemma 6** ([15, 16]). *There exist $O(1)$-approximation algorithms for* Round-UFPP *where all flows are $\frac{1}{4}$-small.*

Algorithm 1: `ProcMids`

input : A set of $[\frac{1}{4}, \alpha]$-mid flows F
output : A partition of F into rounds (colors)

```
for i ← 1 to ⌈log c_max⌉ do
    F_i ← {f_k ∈ F | 2^{i-1} ≤ b_k < 2^i};
    (C_1^i, C_2^i) ← FlowDec(F_i);
R ← ColOptimize({(C_1^k, C_2^k) : k = 1, ..., ⌈log c_max⌉});
return R;
```

However, these results do not extend to the case where α is an arbitrary constant in $(0, 1)$. In contrast, we present an algorithm that works for any choice of $\alpha \in (0, 1)$. In our algorithm, flows are partitioned according to the ratio of their size to their bottleneck capacity. If $\alpha \leq 1/4$, we simply use Lemma 6. Suppose that $\alpha > 1/4$. The overall idea is to further partition the set of flows into two subsets and solve each independently. This motivates the following definition.

▶ **Definition 7.** Given two real numbers $0 \leq \beta < \alpha < 1$, a flow f_i is said to be $[\beta, \alpha]$-*mid* if $\sigma_i \in [\beta \cdot b_i, \alpha \cdot b_i]$. Accordingly, we define the corresponding set of flows as

$$F^M(\beta, \alpha) = \{f \in F \mid f \text{ is } [\beta, \alpha]\text{-mid}\}.$$

Observe that, $F^M(\beta, \alpha) = F^S_\alpha \cap F^L_\beta$.

In the remainder of this section, we present an $O(1)$-approximation algorithm, called `ProcMids`, for $F^M(1/4, \alpha)$. `ProcMids` (see Algorithm 1) starts by partitioning $F^M(1/4, \alpha)$ into $\lceil \log c_{\max} \rceil$ classes according to their bottleneck capacity.

Next, it computes a coloring for each class by running a separate procedure called `FlowDec`, explained in §3.1. This will result in a coloring of $F^M(1/4, \alpha)$ using $O(r_{\max} \log c_{\max})$ colors. Finally, `ProcMids` runs `ColOptimize`, described in §3.2, to optimize color usage in different subsets; this results in the removal logarithmic factor and, thereby, a more efficient coloring using only $O(r_{\max})$ colors.

3.1 A logarithmic approximation

Procedure `FlowDec` partitions F^M_ℓ into $O(r_{\max}(F^M_\ell))$ rounds. In each iteration, it calls procedure `rCover` (Algorithm 2) which takes as input a subset $F'_\ell \subseteq F^M_\ell$ and returns two disjoint feasible subsets C_1, C_2 of F'_ℓ. In other words, flows in each subset can be scheduled simultaneously without causing any capacity violation. On the other hand, these two subsets cover all the links used by the flows in F'_ℓ. More formally, C_1 and C_2 are guaranteed to have the following two properties:

(P1) $\forall e \in E : |C_1(e)| \leq 1$ and $|C_2(e)| \leq 1$,
(P2) $|F'_\ell(e)| > 1 \Rightarrow C_1(e) \cup C_2(e) \neq \emptyset$.

`rCover` maintains a set of flows F'' which is initially empty. It starts by finding the longest flow f_{i_1} among those having the first (leftmost) source node. Next, it processes the flows in a loop. In each iteration, the procedure looks for a flow overlapping with the currently selected flow f_{i_k}. If one is found, it is added to the collection and becomes the current flow. Otherwise, the next flow is chosen among those remaining flows that start after the current flow's sink t_{i_k}. Finally, `rCover` splits F'' into two feasible subsets and returns them.

Algorithm 2: `rCover`

input : A set of flows F
output : Two disjoint feasible subsets of F satisfying Properties (P1) and (P2)

$F'' \leftarrow \emptyset$;
$s_{\min} \leftarrow \min_{f_k \in F} s_k$;
$t_{i_1} \leftarrow \max\{t_k \mid f_k \in F \text{ and } s_k = s_{\min}\}$;
$F'' \leftarrow \{f_{i_1}\}$;
$F \leftarrow F \backslash \{f_{i_1}\}$;
$k \leftarrow 1$;
while $t_{i_k} < \max_{f_i \in F}\{t_i\}$ **do**
  if $\exists f_i \in F : s_i \leq t_{i_k}$ *and* $t_i > t_{i_k}$ **then**
    $t_{i_{k+1}} \leftarrow \max\{t_i \mid f_i \in F \text{ and } s_i \leq t_{i_k}\}$;
  else
    $s_{\min} \leftarrow \min\{s_i \mid f_i \in F,\ s_i > t_{i_k}\}$;
    $t_{i_{k+1}} \leftarrow \max\{t_i \mid f_i \in F,\ s_i = s_{\min}\}$;
  $F'' \leftarrow F'' \cup \{f_{i_{k+1}}\}$;
  $k \leftarrow k + 1$;
$C_1 \leftarrow \{f_{i_j} \in F'' \mid j \text{ is odd}\}$;
$C_2 \leftarrow \{f_{i_j} \in F'' \mid j \text{ is even}\}$;
return (C_1, C_2);

Algorithm 3: `ColOptimize`

input : A set of pairs $\{(C_1^i(j), C_2^i(j))\}$, parameter τ
output : A new set of pairs $\{(D_1^i(j), D_2^i(j))\}$

for $i \leftarrow 1$ **to** $4r_{\max}$ **do**
  for $k \leftarrow 1$ **to** τ **do**
    $D_1^i(k) \leftarrow \bigcup_{z=0}^{\lceil (\log c_{\max})/\tau \rceil - 1} C_1^i(z\tau + k)$;
    $D_2^i(k) \leftarrow \bigcup_{z=0}^{\lceil (\log c_{\max})/\tau \rceil - 1} C_2^i(z\tau + k)$;
return $\{(D_1^i(k), D_2^i(k)) : k = 1, ..., \tau \text{ and } i \in \{1, ..., 4r_{\max}\}\}$;

▶ **Lemma 8.** *Procedure* `rCover` *finds two feasible subsets C_1 and C_2 satisfying properties (P1) and (P2).*

▶ **Lemma 9.** *Procedure* `FlowDec` *partitions F_ℓ^M into at most $8r_{\max}(F_\ell^M)$ feasible subsets.*

3.2 Removing the log factor

In this subsection, we illustrate Procedure `ColOptimize` (see Algorithm 3), which removes the logarithmic factor by optimizing color usage. The result is a coloring with $O(r_{\max})$ colors.

Let τ be a constant to be determined later. Intuitively, the idea is to combine subsets of different levels in an alternating manner with τ serving as the granularity parameter. More precisely, let $C_a^i(j)$, where $a \in \{1, 2\}$, $j \in \{1, ..., \lceil \log c_{\max} \rceil\}$, and $i \in \{1, ..., 4r_{\max}\}$, denote the set of colors resulting from the execution of `FlowDec`. `ColOptimize` combines colors from different classes to reduce the number of colors by a factor of $\tau / \lceil \log c_{\max} \rceil$ resulting in $4\tau \cdot r_{\max}$ colors being used. Next, we show that setting $\tau = \log(1/(1-\alpha)) + 2$ results in a valid coloring.

▶ **Lemma 10.** *For $\tau = \log(1/(1-\alpha)) + 2$, the sets $D^i_a(k)$, where $a \in \{1,2\}$, $k \in \{1,...,\tau\}$, and $i \in \{1,...,4r_{\max}\}$, constitute a valid coloring.*

The main result of this section now directly follows from Lemma 10.

▶ **Theorem 11.** *For any $\alpha \in (0,1)$, there exists an offline $O(\log(1/1-\alpha))$-approximation algorithm for* Round-UFPP *with α-small flows. In particular, we have a constant-factor approximation for any constant $\alpha < 1$.*

4 Algorithms for general Round-UFPP instances

In what follows, we present offline and online algorithms for general instances of Round-UFPP. Our treatment of large flows involves a reduction from Round-UFPP to the rectangle coloring problem (RCOL) which is discussed in §4.1. Next, in §4.2, we design an online algorithm for the RCOL instances arising from the reduction. Later, in §4.3, we cover our online algorithm for Round-UFPP with $\frac{1}{4}$-large flows. Finally, in §4.4, we present our final algorithm for the general Round-UFPP instances.

4.1 The reduction from Round-UFPP with large flows to RCOL

▶ **Definition 12** (Rectangle Coloring Problem (RCOL).)**.** Given a collection $\mathcal{R}$ of n axis-parallel rectangles, the objective is to color the rectangles with the minimum number of colors such that rectangles of the same color are disjoint.

Each rectangle $R \in \mathcal{R}$ is given by a quadruple $(x^l(R), x^r(R), y^t(R), y^b(R))$ of real numbers, corresponding to the x-coordinates of its left and right boundaries and the y-coordinates of its top and bottom boundaries, respectively. More precisely, $R = \{(x,y) \mid x^l(R) \le x \le x^r(R)$ and $y^b(R) \le y \le y^t(R)\}$. When the context is clear, we may omit R and write x^l, x^r, y^t, y^b. Two rectangles R and R' are called compatible if they do not intersect each other; else, they are called incompatible.

The reduction from Round-UFPP with large flows to RCOL is based on the work in [10]. It starts by associating with each flow $f_i = (s_i, t_i, \sigma_i)$, a rectangle $R_i = (s_i, t_i, b_i, b_i - \sigma_i)$. If we draw the capacity profile over the path P, then R_i is a rectangle of thickness σ_i sitting under the curve touching the "ceiling." Let $\mathcal{R}(F)$ denote the set of rectangles thus associated with flows in F. We assume, without loss of generality, that rectangles do not intersect on their border; that is, all intersections are with respect to internal points. We begin with an observation stating that a disjoint set of rectangles constitutes a feasible set of flows.

▶ **Observation 13** ([10])**.** *Let $\mathcal{R}(F)$ be a set of disjoint rectangles corresponding to a set of flows F. Then, F is a feasible set of flows.*

The main result here is that if all flows in F are k-large then an optimal coloring of $\mathcal{R}(F)$ is at most a factor of $2k$ worse than the optimal solution to Round-UFPP instance arising from F. The following key lemma is crucial to the result.

▶ **Lemma 14** ([10])**.** *Let F be a feasible set of flows, and let $k \ge 2$ be an integer, such that every flow in F is $\frac{1}{k}$-large. Then there exists a $2k$ coloring of $\mathcal{R}(F)$.*

As an immediate corollary, we get the following.

▶ **Corollary 15.** *Let F be a feasible set of flows, and let $k \ge 2$ be an integer, such that every flow in F is $\frac{1}{k}$-large. Then, $\chi(\mathcal{R}(F)) \le 2k\chi(F)$.*

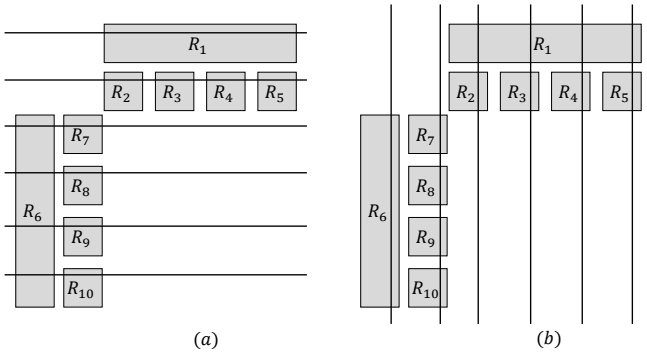

Figure 2 A collection $\mathcal{R}$ of 4-line-sparse rectangles. The lines can be either (a) horizontal or (b) vertical.

Proof. Consider an optimal coloring C of F with $\chi(F)$ colors. Apply Lemma 14 to each color class C_i, for $1 \leq i \leq \chi(F)$, to get a $2k$-coloring of $\mathcal{R}(C_i)$. The final result is a coloring of $\mathcal{R}(F)$ using at most $2k\chi(F)$ colors. ◀

We are ready to state the main result of this subsection.

▶ **Lemma 16.** *Suppose there exists an offline α-approximation (online α-competitive) algorithm $\mathfrak{A}$ for* RCOL*. Then, for every integer $k \geq 2$ there exists an offline $2k\alpha$-approximation (online $2k\alpha$-competitive) algorithm for* Round-UFPP *consisting of $\frac{1}{k}$-large flows.*

Proof. Given a set F of $\frac{1}{k}$-large flows for some integer $k \geq 2$, construct the set of associated rectangles $\mathcal{R}(F)$ and apply the algorithm $\mathfrak{A}$ to it. The solution is a valid Round-UFPP solution (Observation 13). Furthermore, by Corollary 15,

$$\mathfrak{A}(\mathcal{R}(F)) \leq \alpha\chi(\mathcal{R}(F)) \leq 2k\alpha\chi(F).$$

Finally, the reduction does not depend on future flows; hence, it is online in nature. ◀

4.2 Algorithms for RCOL

In this section, we consider algorithms for the rectangle coloring problem (RCOL). We begin by introducing a key notion measuring the sparsity of rectangles with respect to a set of lines. This is similar to the concept of point sparsity investigated by Chalermsook [12].

▶ **Definition 17** (s-line-sparsity)**.** A collection of rectangles $\mathcal{R}$ is *s-line-sparse* if there exists a set of axis-parallel lines $L_{\mathcal{R}}$ (called an *s-line-representative set* of $\mathcal{R}$), such that every rectangle $R \in \mathcal{R}$ is intersected by $k_R \in [1, s]$ lines in $L_{\mathcal{R}}$ (see Figure 2 for an example).

For simplicity, we assume that representative lines are all horizontal. The objective is to design an online $O(\log s)$-competitive algorithm for RCOL consisting of s-line-sparse rectangles. In the online setting, rectangles appear one by one; however, we assume that an s-line-representative set $L_{\mathcal{R}}$ is known in advance. As we will later see, this will not cause any issues since the RCOL instances considered here arise from Round-UFPP instances with large flows from which it is straightforward to compute s-line-representative sets. In the offline case, on the other hand, we get a $\log(n)$ approximation by (trivially) computing an n-line-representative set–associate to each rectangle an arbitrary line intersecting it. The remainder of this subsection is organized as follows. First, in §4.2.1, we consider the 2-line-sparse case. Later, in §4.2.2, we study the general s-line-sparse case.

4.2.1 The 2-line-sparse case

Consider a collection of rectangles $\mathcal{R}$ and a 2-line-representative set $L_{\mathcal{R}} = \{\ell_0, \ell_1, ..., \ell_k\}$ (that is, each rectangle R is intersected by either one or two lines in $L_{\mathcal{R}}$) where the rectangles in $\mathcal{R}$ appears in an online fashion. Recall, however, that the line set $L_{\mathcal{R}}$ is known in advance. Without loss of generality, assume that $y(\ell_0) < y(\ell_1) < ... < y(\ell_k)$.

For each $R \in \mathcal{R}$, let $T(R)$ denote the index of the topmost line in $L_{\mathcal{R}}$ that intersects R; $T(R) = \max\{i \mid \ell_i \text{ intersects } R\}$. Next, partition $\mathcal{R}$ into three subsets

$$\mathcal{R}_l = \{R \in \mathcal{R} \mid T(R) \equiv l \mod 3\}, \text{ for } l = 0, 1, 2. \tag{2}$$

The following lemma shows that each of the above subsets can be viewed as a collection of interval coloring problem (ICP) instances.

▶ **Lemma 18.** *Suppose two rectangles $R, R' \in \mathcal{R}$ belong to the same subset; that is, $R, R' \in \mathcal{R}_l$ for some $l \in \{0, 1, 2\}$. Then, the following are true.*

(1) *If $T(R) = T(R')$ and the projection of R and R' on the x-axis have a non-empty intersection, then $R \cap R' \neq \emptyset$.*

(2) *If $T(R) \neq T(R')$, then $R \cap R' = \emptyset$.*

We will use the optimal 3-competitive online algorithm due to Kierstead and Trotter for ICP [20]. The algorithm colors an instance of ICP of clique size ω with at most $3\omega - 2$ colors which matches the lower bound shown in the same paper. Henceforth, we refer to this algorithm as the KT algorithm.

Now we can present an $O(1)$-competitive online algorithm, named COL2SP, with a known 2-line-representative set. COL2SP computes a partition of $\mathcal{R}$ into $\mathcal{R}_0, \mathcal{R}_1$, and $\mathcal{R}_2$ as explained above. Then, it applies the KT algorithm to each subset. Note that COL2SP can be seen as executing multiple instances of the KT algorithm in parallel.

▶ **Lemma 19.** *Algorithm COL2SP is an online $O(1)$-competitive algorithm for RCOL on 2-line-sparse instances given prior knowledge of a 2-line-representative set for the incoming rectangles. Moreover, COL2SP uses at most $3 \cdot \omega(\mathcal{R})$ colors.*

4.2.2 The s-line-sparse case

Consider a set of s-line-sparse rectangles $\mathcal{R}$ and an s-line-representative set $L_{\mathcal{R}}$. Our goal in this subsection is to demonstrate a partitioning of $\mathcal{R}$ into $O(\log s)$ 2-line-sparse subsets, where each subset is accompanied by its own 2-line-representative set.

Given a set of lines L, we define the degree of a rectangle $R \in \mathcal{R}$, with respect to L, to be the number of lines in L that intersect R,

$$\text{Deg}_L(R) = |\{\ell \in L \mid \ell \cap R \neq \emptyset\}|.$$

We say that a rectangle $R \in \mathcal{R}$ is of level $l \geq 0$ with respect to $L_{\mathcal{R}}$, if $2^l \leq \text{Deg}_{L_{\mathcal{R}}}(R) < 2^{l+1}$. The partitioning is based on the level of rectangles. More precisely, $\mathcal{R}$ is partitioned into $\lceil \log s \rceil + 1$ "levels"

$$\text{Lev}(i) = \{R \in \mathcal{R} \mid R \text{ is of level } i\}, \text{ for } i = 0, 1, ..., \lceil \log s \rceil.$$

Next we show that each level is a 2-line-sparse set. To this end, we present a 2-line-representative set for each level. Let $L_{\mathcal{R}} = \{\ell_1, \ell_2, ..., \ell_k\}$ and define

$$S(i) = \{\ell_j \in L_{\mathcal{R}} \mid j \equiv 1 \mod 2^i\}, \text{ for } i \in \{0, ..., \lceil \log s \rceil\}.$$

Algorithm 4: `RectCol`

input : A rectangle $R \in \mathcal{R}$
input : The last state of `RectCol`; an s-representative-line set $L_{\mathcal{R}}$ for $\mathcal{R}$
output : A color for R

$i \leftarrow \text{argmin}_j(2^j \leq \text{Deg}_{L_{\mathcal{R}}}(R) < 2^{j+1})$;
$\text{Lev}(i) \leftarrow \text{Lev}(i) \cup \{R\}$;
return `COL2SP`$(R, L_{\mathcal{R}})$;

▶ **Lemma 20.** *For every $i \in \{0, ..., \lceil \log s \rceil\}$, $\text{Lev}(i)$ is a 2-line-sparse set and $S(i)$ is a 2-line-representative set for $\text{Lev}(i)$.*

We are ready to present an $O(\log s)$-competitive online algorithm, named `RectCol`, for RCOL with a known line-representative set. Algorithm `RectCol` works as follows (see Algorithm 4).

▶ **Lemma 21.** `RectCol` *is an online $O(\log s)$-competitive algorithm for* RCOL *with s-line-sparse rectangles, given a representative-line set. Moreover,* `RectCol` *uses $O(\omega(\mathcal{R}) \cdot \log s)$ colors.*

4.3 An algorithm for Round-UFPP with large flows

We are ready to present `ProcLarges`, an algorithm for Round-UFPP with large flows. For concreteness, we present the algorithm for $\frac{1}{4}$-large flows; this result can be easily generalized to α-large flows for any $\alpha \leq 1/2$.

The online algorithm we have designed for RCOL need to have access to an s-line-representative set $L_{\mathcal{R}}$ for the set of rectangles $\mathcal{R}$. In our case, these rectangles are constructed from flows (§4.1) which themselves arrive in an online fashion. However, all we need to be able to compute an s-line-representative set is the knowledge of the path over which the flows will be running–that is $P = (V, E)$ with capacities $\{c_e\}_{e \in E}$ (recall that we assume that $c_{\min} = 1$, which can always be achieved via scaling if needed). It is possible to construct (at least) three different s-line-representative sets for $\mathcal{R}$:

L_1 A set of $s = \lceil \log_{4/3} c_{\max} \rceil + 1$ horizontal lines $L = \{l_0, l_1, ..., l_s\}$ where the y-coordinate of the ith line is $y(l_i) = (3/4)^i \cdot c_{\max}$. Note that ℓ_0 is the topmost line.
L_2 A set of m vertical lines, one per edge in the path.
L_3 A set of n axis-parallel lines, one per rectangle.

Note that L_3 is only useful in the offline setting. It is obvious that L_2 and L_3 are valid line-representative sets for $\mathcal{R}$. Below, we show that L_1 is valid as well.

▶ **Lemma 22.** *L_1 is an s-line-representative set for $\mathcal{R}(\mathcal{F})$.*

▶ **Theorem 23.** `ProcLarges` *is an $O(\log \log c_{\max})$-competitive algorithm for* Round-UFPP *with $\frac{1}{4}$-large flows. Furthermore, the bound can be improved to $O(\min(\log m, \log \log c_{\max}))$.*

Proof. `ProcLarges` executes algorithm `RectCol` on $\mathcal{R}(F)$ with a representative-line set $L = L_1$ of size $O(\log c_{\max})$. The colors returned by `RectCol` are used for the flows without modification. Now, setting $s = O(\log c_{\max})$, Lemma 21 states that Algorithm `RectCol` uses $O(\omega(\mathcal{R}(F)) \log \log c_{\max})$ colors. Lemma 16 completes the argument. Finally, note that running algorithm `RectCol` with $L = L_2$ as the representative-line set, we get a sparsity of $s = m$ and a coloring using $O(\omega(\mathcal{R}(F)) \log m)$ colors. To get the improved bound, we run the algorithm with $L = L_1$, if $\log c_{\max} \leq m$; else, we run it with $L = L_2$. ◀

4.4 Putting it together – The final algorithm

At this point, we have all the ingredients needed to present our final algorithm, `SolveRUFPP`, for Round-UFPP. `SolveRUFPP` simply uses procedure `ProcLarges` (§4.3) for $\frac{1}{4}$-large flows and procedure `ProcSmalls` for $\frac{1}{4}$-small flows. For `ProcSmalls`, we can use our algorithm in §3 or the 16-competitive algorithm in [15] in the offline case; and the 32-competitive algorithm in [16] in the online case.

▶ **Theorem 24.** *There exists an online $O(\min(\log m, \log\log c_{\max}))$-competitive algorithm and an offline $O(\min(\log n, \log m, \log\log c_{\max}))$-approximation algorithm for* Round-UFPP.

Proof. In the online case, `ProcSmalls` is a 32-competitive [16]. On the other hand, by Proposition 23, `ProcLarges` is an $O(\min(\log m, \log\log c_{\max}))$-competitive. Thus overall, algorithm `SolveRUFPP` is $O(\min(\log m, \log\log c_{\max}))$-competitive. In the offline case, since the set of flows $\mathcal{F}$ is known in advance, we can get a slightly better bound by using L_3 in §4.3 as the third line-representative set (of sparsity $s = n$). Thus we get the $O(\min(\log n, \log m, \log\log c_{\max}))$ bound by running the algorithm three times with L_1, L_2, and L_3 and using the best one. ◀

5 Concluding remarks

In this paper, we present improved offline approximation and online competitive algorithms for Round-UFPP. Our work leaves several open problems. First, is there an $O(1)$-approximation algorithm for offline Round-UFPP? Second, can we improve the competitive ratio achievable in the online setting to match the lower bound of $\Omega(\log\log\log c_{\max})$ shown in [16], or improve the lower bound? From a practical standpoint, it is important to analyze the performance of simple online algorithms such as First-Fit and its variants for Round-UFPP and RCOL. Another natural direction for future research is the study of Round-UFP and variants on more general graphs.

References

1. Udo Adamy and Thomas Erlebach. Online coloring of intervals with bandwidth. In Klaus Jansen and Roberto Solis-Oba, editors, *Approximation and Online Algorithms, First International Workshop, WAOA 2003, Budapest, Hungary, September 16-18, 2003, Revised Papers*, volume 2909 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 2003. doi:10.1007/978-3-540-24592-6_1.
2. Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. A mazing 2+eps approximation for unsplittable flow on a path. *CoRR*, abs/1211.2670, 2012.
3. Esther M. Arkin and Ellen B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1):1 – 8, 1987.
4. E. Asplund and B. Grünbaum. On a coloring problem. *Mathematica Scandinavica*, 8(0):181–188, 1960.
5. Yossi Azar, Amos Fiat, Meital Levy, and N. S. Narayanaswamy. An improved algorithm for online coloring of intervals with bandwidth. *Theor. Comput. Sci.*, 363(1):18–27, 2006. doi:10.1016/j.tcs.2006.06.014.
6. Nikhil Bansal, Amit Chakrabarti, Amir Epstein, and Baruch Schieber. A quasi-ptas for unsplittable flow on line graphs. In Jon M. Kleinberg, editor, *Proceedings of the 38th Annual ACM Symposium on Theory of Computing, Seattle, WA, USA, May 21-23, 2006*, pages 721–729. ACM, 2006. doi:10.1145/1132516.1132617.

7 Nikhil Bansal, Zachary Friggstad, Rohit Khandekar, and Mohammad R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In Claire Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 702–709. SIAM, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496847.

8 Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, September 2001.

9 Mark Bartlett, Alan M. Frisch, Youssef Hamadi, Ian Miguel, Armagan Tarim, and Chris Unsworth. The temporal knapsack problem and its solution. In Roman Barták and Michela Milano, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings*, volume 3524 of *Lecture Notes in Computer Science*, pages 34–48. Springer, 2005. doi:10.1007/11493853_5.

10 P. Bonsma, J. Schulz, and A. Wiese. A constant factor approximation algorithm for unsplittable flow on paths. In *FOCS'11*, pages 47–56, 2011.

11 Gruia Calinescu, Amit Chakrabarti, Howard Karloff, and Yuval Rabani. An improved approximation algorithm for resource allocation. *ACM Trans. Algorithms*, 7(4):48:1–48:7, September 2011.

12 Parinya Chalermsook. Coloring and maximum independent set of rectangles. *APPROX'11*, pages 123–134, 2011. URL: http://dx.doi.org/10.1007/978-3-642-22935-0_11.

13 Chandra Chekuri, Marcelo Mydlarz, and F. Bruce Shepherd. Multicommodity demand flow in a tree. In Jos C. M. Baeten, Jan Karel Lenstra, Joachim Parrow, and Gerhard J. Woeginger, editors, *ICALP'03*, pages 410–425, 2003.

14 Andreas Darmann, Ulrich Pferschy, and Joachim Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411(49):4217 – 4234, 2010.

15 Khaled M. Elbassioni, Naveen Garg, Divya Gupta, Amit Kumar, Vishal Narula, and Arindam Pal. Approximation algorithms for the unsplittable flow problem on paths and trees. In Deepak D'Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 267–275. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.267.

16 Leah Epstein, Thomas Erlebach, and Asaf Levin. Online capacitated interval coloring. *SIAM Journal on Discrete Mathematics*, 23(2):822–841, 2009.

17 Leah Epstein and Meital Levy. Online interval coloring and variants. In *ICALP'05*, pages 602–613, 2005.

18 N. Garg, V. V. Vazirani, and M. Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997.

19 H. A. Kierstead. The linearity of first-fit coloring of interval graphs. *SIAM Journal on Discrete Mathematics*, 1(4):526–530, 1988.

20 H. A. Kierstead and W. T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

21 Alexandr Kostochka. Coloring intersection graphs of geometric figures with a given clique number. In *Contemporary Mathematics 342, AMS*, 2004.

22 Cynthia A. Phillips, R. N. Uma, and Joel Wein. Off-line admission control for general scheduling problems. In *Journal of Scheduling*, pages 879–888, 2000.