Algorithmica (2023) 85:563–583
https://doi.org/10.1007/s00453-022-01043-6

Improved Algorithms for Scheduling Unsplittable Flows on Paths

Hamidreza Jahanjou[1] · **Erez Kantor**[2] · **Rajmohan Rajaraman**[3]

Received: 21 December 2018 / Accepted: 20 September 2022 / Published online: 1 October 2022

Abstract

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 size on a given path with heterogeneous edge capacities. Round-UFPP is known to be NP-hard and there are constant-factor approximation algorithms under the no bottleneck assumption (NBA), which stipulates that maximum size of any flow is at most the minimum global edge capacity. In this work, we present improved online and offline algorithms for Round-UFPP *without the NBA*. 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. Next, our main result is an online $O(\log\log c_{\max})$-competitive algorithm for Round-UFPP where $c_{\max}$ is the largest edge capacity, improving upon the previous best bound of $O(\log c_{\max})$ due to Epstein et al. (SIAM J Discrete Math 23(2):822–841, 2009). These new results lead 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.

Keywords Unsplittable flows · Optical routing · Scheduling · Interval scheduling

A previous version of this paper has appeared in the proceedings of ISAAC 2017.

✉ Hamidreza Jahanjou
h.jahanjou@northeastern.edu

Erez Kantor
ekantor@akamai.com

Rajmohan Rajaraman
r.rajaraman@northeastern.edu

1 Northeastern University, San Jose, CA, USA

2 Akamai Technologies Inc., Cambridge, MA, USA

3 Northeastern University, Boston, MA, USA

1 Introduction

The *unsplittable flow problem on paths* (UFPP) considers selecting a maximum-weight subset of flows to be routed simultaneously over a path subject to 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 each round collectively 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 capacity constraints. The objective is to partition $\mathcal{F}$ into the smallest number of feasible sets (rounds or colors) $R_1, \ldots, R_t$.

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 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 Sect. 1.1). A major breakthrough was the design of $O(1)$-approximation algorithms for the unsplittable flow problem on paths (UFPP) without the NBA [2, 11]. In this paper, we make progress towards an optimal algorithm for Round-UFPP *without* imposing the 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 *a priori*. Moreover, every flow must be scheduled (i.e. assigned to a round) 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 whose capacity equals the bin size). 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 [3, 7–10, 12, 14, 15, 26]. Recently, $O(1)$-approximation algorithms were discovered for UFPP (without NBA) [2, 11]. 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 [19].

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 online algorithm for ICP with a competitive ratio of 3. Their algorithm uses at most $3\omega - 2$ colors, where ω denotes the maximum clique size [22]. Observe that, since interval graphs are perfect, 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 [6, 18] and a lower bound of slightly greater than 3 is known [21]. The online Round-UFPP is considered in Epstein et al. [17]. 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 lower bounds of $\Omega(\log \log m)$ and $\Omega(\log \log \log \frac{c_{\max}}{c_{\min}})$ on the competitive ratio achievable for Round-UFPP. We note that in [17], parameters m and n represent the number of flows and edges respectively. In the offline setting, a 24-approximation algorithm for Round-UFPP-NBA is presented in [16].

A relevant research topic is optical routing and scheduling (also known as path coloring) which corresponds to Round-UFP[2] where all flows and edge capacities are 1. This problem has been studied in [5, 24, 25, 27] among others. It is worthwhile to note that, path coloring is hard to approximate to within a factor of $n^{1-\epsilon}$, for all $\epsilon > 0$, even on grids [20, 28].

1.2 Our Results

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

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

[2] Not to be confused with Round-UFPP where the graph is restricted to be a path.

algorithm and an online $O(\min(\log m, \log\log c_{\max}))$-competitive algorithm for Round-UFPP. Note that $c_{\min} = 1$ without loss of generality.

Our algorithm for large instances, which improves on the previous $O(\log c_{\max})$-bound [17], is based on a reduction to the classic rectangle coloring problem (c.f. [4, 13, 23]). In particular, we introduce the notion of "line-sparse" rectangles that may be of independent interest, and show how competitive algorithms for coloring this class of rectangles lead to competitive algorithms for Round-UFPP.

2 Preliminaries

In Round-UFPP we are given a path P and a set of flows $\mathcal{F}$. The path P consists of m links and $m+1$ vertices, enumerated left-to-right as $v_0, e_1, v_1, \ldots, v_{m-1}, e_m, v_m$ with link capacities $\{c_j\}_{j\in[m]}$. The set of flows $\mathcal{F}$ consists of n flows $\{f_i = (s_i, t_i, \sigma_i) : i \in [n]\}$, where $s_i < t_i$ represent the two endpoints of flow f_i, and σ_i denotes the size of the flow (note the slight abuse of notation: $s_i < t_i$ means that s_i is to the left of t_i). We say that a flow f_i, with endpoints $s_i = v_k$ and $t_i = v_l$, uses a link e_j if $k < j \leq l$.

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 edges are called the bottleneck edges of f_i (a flow can have multiple bottleneck edges). Additionally, an edge is called a bottleneck edge if it is a bottleneck edge for some flow.

Definition 2 A set of flows R is called *feasible* if all of its members can be routed simultaneously without causing capacity violation.

The objective in Round-UFPP is to partition $\mathcal{F}$ into the smallest number of feasible sets $R_1, \ldots, R_t$. A feasible set is also referred to as a *round*. Alternatively, partitioning can be seen as coloring where rounds correspond to colors.

Definition 3 For a set of flows F, we define its *chromatic number*, $\chi(F)$, to be smallest number of colors (rounds) into which F can be partitioned.

Definition 4 The *congestion* of an edge e_j with respect to a set of flows F is

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

where $F(j)$ is the subset of flows in F using edge e_j. In other words, $r_j(F)$ is the ratio of the total size of flows in F using e_j to its capacity. 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 $r_{\max}$.

Remark 1 When referring to an arbitrary edge e, we drop the index and write c_e and $r_e(F)$ in place of c_j and $r_j(F)$ respectively.

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

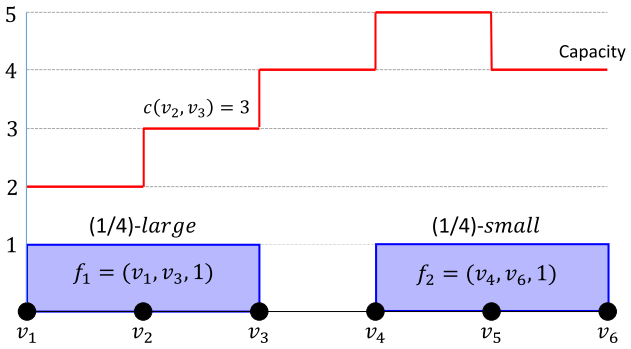

Fig. 1 An example of a path with 5 links and two flows. The first flow f_1 is from v_1 to v_3 of size 1; the second flow f_2 is from v_4 to v_6 also of size 1. Even though both flows have the same size, f_1 is $\frac{1}{4}$-large whereas f_2 is $\frac{1}{4}$-small. The reason is different bottleneck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

Observation 1 $\chi(\mathcal{F}) \geq \lceil r_{\max}(\mathcal{F}) \rceil$.

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 $c_{\min} = 1$ (we can always rescale if needed). 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 Fig. 1 for an example). Accordingly, a set of flows F is divided into small and large classes

$$F_\alpha^S = \{f \in F \mid f \text{ is } \alpha\text{-small}\}; \quad F_\alpha^L = \{f \in F \mid f \text{ is } \alpha\text{-large}\}. \tag{2}$$

As is often the case for unsplittable flow algorithms, we treat small and large instances independently. In Sects. 3 and 4 we present algorithms for Round-UFPP with α-small and α-large instances respectively. The exact value of α will be set later.

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 [16, 17] respectively.

Theorem 2 [16, 17] *There exist $O(1)$-approximation algorithms for instances of Round-UFPP where all flows are $\frac{1}{4}$-small.*

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 Theorem 2. On the other hand, if $\alpha > 1/4$, our approach is to further partition the set of flows based on their bottleneck capacity. This motivates the following definition.

Definition 6 Given two real numbers $0 \leq \alpha < \beta < 1$, a flow f_i is said to be $(\alpha, \beta]$-*mid* if $\sigma_i \in (\alpha \cdot b_i, \beta \cdot b_i]$. Accordingly, we define the corresponding set of flows as

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

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

In the remainder of this section, we present an $O(1)$-approximation algorithm, called `ProcMids`, for $F^M(1/4, \alpha)$. `ProcMids` (Algorithm 1) initially computes a less efficient schedule (coloring) and then improves on it by reusing rounds (colors). Specifically, the algorithm works in three steps. First, it starts by partitioning $F^M(1/4, \alpha)$ into $\lceil \log c_{\max} \rceil$ *classes* according to their bottleneck capacity

$$F^M_\ell = \begin{cases} \{f_i \in F^M(1/4, \alpha) \mid 1 \leq b_i \leq 2^\ell\}, & \text{if } \ell = 1 \\ \{f_i \in F^M(1/4, \alpha) \mid 2^{\ell-1} < b_i \leq 2^\ell\}, & \text{if } \ell = 2, \ldots, \lceil \log c_{\max} \rceil \end{cases} \tag{3}$$

In words, F^M_ℓ is the subset of flows in $F^M(1/4, \alpha)$ whose bottleneck capacity fall into the bracket $(2^{\ell-1}, 2^\ell]$ (for $\ell = 1$ we use $[2^{\ell-1}, 2^\ell]$ instead). The following implication holds for every class F^M_ℓ

$$\forall f_i \in F^M_\ell \Rightarrow 2^{\ell-3} \leq \sigma_i \leq \alpha \cdot 2^\ell. \tag{4}$$

Algorithm 1: `ProcMids`

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

1 $F_1 \leftarrow \{f_k \in F \mid 1 \leq b_k \leq 2\}$;
2 $R_1 \leftarrow$ `FlowDec`(F_1);
3 **for** $i \leftarrow 2$ **to** $\lceil \log c_{\max} \rceil$ **do**
4 $\quad F_i \leftarrow \{f_k \in F \mid 2^{i-1} < b_k \leq 2^i\}$;
5 $\quad R_i \leftarrow$ `FlowDec`(F_i);
6 $R' \leftarrow$ `ColOptimize`$(\{\bigcup_{k=1}^{\lceil \log c_{\max} \rceil} R_k\})$;
7 **return** R';

Next, the second step is to compute a coloring for each class by running a separate algorithm called `FlowDec`, explained in Sect. 3.1. This will result in a coloring of $F^M(1/4, \alpha)$ using $O(r_{\max} \log c_{\max})$ colors. Finally, in step 3, `ProcMids` runs

`ColOptimize`, described in Sect. 3.2, to optimize color usage in different levels; this results in removal of the logarithmic factor and, thereby, a more efficient coloring using $O(r_{\max})$ colors.

3.1 A Logarithmic Approximation

The job of `FlowDec` (Algorithm 2) is to schedule a set of flows F into $O(r_{\max}(F))$ rounds (i.e. feasible sets of flows). In each iteration, it extracts from F two disjoint rounds C_1, C_2 and removes them from F. This continues as long as $F \neq \emptyset$.

Algorithm 2: `FlowDec`

input : A set of flows F
output: A partition of F into feasible subsets

```
1 i ← 1;
2 while F ≠ ∅ do
3 |   (C_1^i, C_2^i) ← rCOVER(F);
4 |   F ← F\(C_1^i ∪ C_2^i);
5 |   i ← i + 1;
6 return {(C_1^j, C_2^j) : 1 ≤ j < i};
```

The task of extraction is handled by `rCover` (Algorithm 3) which guarantees that each returned pair (C_1, C_2) has the following properties

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

In words, an edge an used by at most one flow in each subset. Moreover, these two subsets cover all the links used by the flows in F. Recall that $X(e)$ denotes the subset of flows in X that use link e. These two properties suffice to show the following.

Lemma 1 *For all $\ell \in \{1, 2, \ldots, \lceil \log c_{\max} \rceil\}$,* `FlowDec` *partitions F_ℓ^M into at most $2\lceil 8 r_{\max}(F_\ell^M) \rceil$ feasible subsets.*

Proof Let F^i denote the set F at the end of the i-th iteration. Before the first iteration, $F^0 = F_\ell^M$. Consider the set B of bottleneck edges used by flows in F_ℓ^M. By definition, the capacity of each edge in B is in $(2^{\ell-1}, 2^\ell]$ if $\ell > 1$, or $[2^{\ell-1}, 2^\ell]$ if $\ell = 1$. We claim that, in each iteration, the congestion of every edge in B is reduced by at at least $1/8$. Indeed, let $e \in B$ be a bottleneck edge with $r_e(F^{i-1}) > 0$ at the start of the i-th iteration. Since $r_e(F^{i-1}) > 0$, by (P2), e is covered by at least one flow in $C_1^i \cup C_2^i$. Let f_k be such a flow. Then,

$$r_e(F^{i+1}) = r_e(F^i \backslash (C_1^i \cup C_2^i)) \leq r_e(F^i) - \frac{\sigma_k}{c_e} \leq r_e(F^i) - \frac{2^{\ell-3}}{2^\ell} = r_e(F^i) - \frac{1}{8} \tag{5}$$

since, by (4), the size of each flow $f \in F_\ell^M$ is at least $2^{\ell-3}$. Hence, the congestion of each bottleneck edge must have reduced to 0 after at most $\lceil 8r_B(F_\ell^M)\rceil$ iterations, where $r_B(F)$ is the maximum congestion among bottleneck edges with respect to the flows in F; that is, $r_B(F) = \max_{e\in B} r_e(F)$. At the end of the last iteration, the congestion of *all* edges must be 0 since every flow in F_ℓ^M must use an edge in B. Finally, since two feasible subsets are produced in every iteration, the total number of feasible subsets is at most $2\lceil 8r_B(F_\ell^M)\rceil \le 2\lceil 8r_{\max}(F_\ell^M)\rceil$. □

Now we describe `rCover` (Algorithm 3). It maintains a set of flows F' which is initially empty and a current flow f_g. The algorithm processes the flows in the order of increasing left point (i.e. the source vertex). Before the first iteration, the current flow, f_g, is chosen to be the longest among the flows with the leftmost source vertex. At each iteration, `rCover` looks for the longest flow that overlaps with the current flow f_g. If found, it becomes the next current flow $f_{g'}$. If no such flow is found, the next current flow $f_{g'}$ is chosen to be the longest among the flows whose source vertex is immediately to the right of the current flow's sink vertex t_g. Thus in the first case, the longest overlapping flow is chosen and in the second case the longest non-overlapping coming next is chose. In either case, the chosen flow will also be added to F' and removed from further consideration. Finally, `rCover` splits F' into two feasible subsets based on their index and returns them.

Lemma 2 *Procedure* `rCover` *computes two feasible subsets C_1 and C_2 satisfying properties (P1) and (P2).*

Proof Let $F' = \{f_{j_1}, f_{j_2}, \ldots, f_{j_p}\}$ denote the set of flows obtained by `rCover` after termination of the loop, where f_{j_k} is the current flow at iteration k. We first establish that

$$\forall k \in \{1, 2, \ldots, p-1\} : t_{j_k} < t_{j_{k+1}} \tag{6}$$

which follows immediately from the selection of $f_{j_{k+1}}$ in iteration k. There are two possible cases. In the first case, where an overlapping flow is found, $f_{j_{k+1}}$ satisfies $t_{j_{k+1}} > t_{j_k}$ by definition. In the second case, where no overlapping flow is found, $f_{j_{k+1}}$ satisfies $t_{j_{k+1}} > s_{j_{k+1}} > t_{j_k}$ again by definition.

We next show that

$$\forall k \in \{1, 2, \ldots, p-2\} : s_{j_{k+2}} > t_{j_k} \tag{7}$$

which implies that no two flows in C_1 (resp., C_2) overlap, establishing property (P1). To this end, let k be the smallest index for which the above condition is violated. Consider iteration k. In this iteration $f_{j_{k+1}}$ is selected either as the longest flow overlapping with f_{j_k} or as the longest flow not overlapping with f_{j_k}. In the latter case, $t_{j_k} < s_{j_{k+1}}$ and (7) follows immediately. In the former case, we need to consider iteration $k+1$ where $f_{j_{k+2}}$ is chosen. If $t_{j_k} < s_{j_{k+2}}$, there is nothing to prove. Otherwise, not only does $f_{j_{k+2}}$ overlap with f_{j_k} but it is also the longer one due to (6). But this contradicts the fact that `rCover` always selects the longest overlapping flow.

Algorithm 3: `rCover`

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

$F' \leftarrow \emptyset$;
$r_{\max} \leftarrow \max_e r_e(F)$;
$s_{\min} \leftarrow \min_{f_i \in F}\{s_i\}$;
$t_{\max} \leftarrow \max_{f_i \in F}\{t_i\}$;
$g \leftarrow \text{argmax}_i\{t_i \mid f_i \in F \wedge s_i = s_{\min}\}$;
$F' \leftarrow \{f_g\}$;
$F \leftarrow F \backslash \{f_g\}$;
while TRUE **do**
  if $\exists f_i \in F : s_i \leq t_g \wedge t_i > t_g$ **then**
    $g' \leftarrow \text{argmax}_i\{t_i \mid f_i \in F \wedge s_i \leq t_g\}$;
  else
    if $\exists f_i \in F : s_i > t_g$ **then**
      $s_{\min} \leftarrow \min\{s_i \mid f_i \in F,\ s_i > t_g\}$;
      $g' \leftarrow \text{argmax}_i\{t_i \mid f_i \in F,\ s_i = s_{\min}\}$;
    else
      Break `// from the while loop`
  $F' \leftarrow F' \cup \{f_{g'}\}$;
  $F \leftarrow F \backslash \{f_{g'}\}$;
  $g \leftarrow g'$;
`// Define` f_{i_j} `as the` j`-th flow added to` F'.
$C_1 \leftarrow \{f_{i_j} \in F' \mid j \text{ is odd}\}$;
$C_2 \leftarrow \{f_{i_j} \in F' \mid j \text{ is even}\}$;
return (C_1, C_2);

It remains to establish property (P2). The proof is again by contradiction. Let e be the left-most edge for which (P2) is violated, and let f_b be a flow that uses edge e. We consider two cases. The first case is where there exists an index k such that $s_b \leq t_{j_k}$. Let k be the smallest such index. In this case, $t_b > t_{j_k}$, since otherwise, if $t_b \leq t_{j_k}$, then either $s_b < t_b \leq s_{j_k} < t_{j_k}$ or $s_b \leq s_{j_k} \leq t_b \leq t_{j_k}$. In the former, k is not the smallest index where $s_b \leq t_{j_k}$. In the latter, e is already covered by the flow f_{j_k}. Both lead to contradiction. Therefore, the first "if" statement in the loop evaluates to true (i.e. there is a flow which overlaps the flow f_g chosen in the previous iteration). At this point, the algorithm will choose the longest flow overlapping f_g. Consequently, the chosen flow $f_{g'} = f_{j_{k+1}}$ satisfies $t_{g'} \geq t_b$, implying that e is covered by flow $f_{j_{k+1}}$, leading to a contradiction. The second case is where there is no index k such that $s_b \leq t_{j_k}$; in particular $s_b > t_{j_p}$ where f_{j_p} is the last flow in $C_1 \cup C_2$. This leads to another contradiction since the termination condition implies $t_{j_p} \geq t_b$. This establishes property (P2) and completes the proof of the lemma. □

3.2 Removing the log Factor

In this subsection, we present `ColOptimize` (Algorithm 4), which removes the logarithmic factor by combining colors from across classes. The result is a coloring

with $O(r_{\max})$ colors. The algorithm uses a granularity parameter τ which will be set later.

Recall that we started with a set $F^M = F^M(1/4, \alpha)$ of flows which are both 1/4-large and α-small. Next, we partitioned F^M into $\lceil \log c_{\max} \rceil$ classes based on their bottleneck capacity. Subsequently, the execution of `FlowDec` on each class F^M_ℓ results in at most $\lceil 8r_{\max}(F^M_\ell) \rceil$ pairs of feasible subsets. Therefore, since $r_{\max}(F^M_\ell) \leq r_{\max}$, in total we end up with the following set of feasible subsets (rounds).

$$\{C^i_a(\ell) : 1 \leq \ell \leq \lceil \log c_{\max} \rceil, 1 \leq i \leq \lceil 8r_{\max} \rceil, a \in \{1, 2\}\}. \tag{8}$$

Algorithm 4: `ColOptimize`

input : A set of pairs $\{(C^j_1(\ell), C^j_2(\ell))\}$, $1 \leq \ell \leq \lceil \log c_{\max} \rceil$, $1 \leq j \leq \lceil 8r_{\max} \rceil$
input : A parameter τ
output: A set $\{D^t_a(k)\}$, $a \in \{1, 2\}$, $1 \leq k \leq \tau$, $1 \leq t \leq \lceil 8r_{\max} \rceil$

for $t \leftarrow 1$ **to** $\lceil 8r_{\max} \rceil$ **do**
  for $k \leftarrow 1$ **to** τ **do**
    $D^t_1(k) \leftarrow \bigcup_{z=0}^{\lceil (\log c_{\max})/\tau \rceil - 1} C^t_1(z\tau + k)$;
    $D^t_2(k) \leftarrow \bigcup_{z=0}^{\lceil (\log c_{\max})/\tau \rceil - 1} C^t_2(z\tau + k)$;
return $\bigcup_{a,k,t} \{D^t_a(k)\}$;

For a given value of τ, `ColOptimize` reduces the number of colors by a factor of $\tau / \lceil \log c_{\max} \rceil$ resulting in $O(\tau \cdot r_{\max})$ colors being used in total. This is achieved by introducing a new set of colors

$$D^t_a(k) = \bigcup_{z=0}^{\lceil (\log c_{\max})/\tau \rceil - 1} C^t_a(z\tau + k) \tag{9}$$

where $a \in \{1, 2\}$, $k \in \{1, \ldots, \tau\}$, and $t \in \{1, \ldots, \lceil 8r_{\max} \rceil\}$. Notice that the same color is being used across different classes. An example is illustrated in Fig. 2. Lemma 3 shows that setting $\tau = \log(1/(1-\alpha)) + 2$ results in a valid coloring using $2\lceil 8r_{\max} \rceil (2 + \log(\frac{1}{1-\alpha}))$ colors. Of course, all flows are assumed to be α-small.

Lemma 3 *For $\tau = 2 + \lceil \log(1/(1-\alpha)) \rceil$, where $\alpha \in (0, 1)$, the sets $D^t_a(k)$, where $a \in \{1, 2\}$, $k \in \{1, \ldots, \tau\}$, and $t \in \{1, \ldots, \lceil 8r_{\max} \rceil\}$, constitute a valid coloring.*

Proof Fix an edge $e \in E$. Let $F(e)$ be the set of flows in F that use e. We need to show that for all valid values of a, k, and t, $D^t_a(k)$ respect the capacity of e. Let $L(e) = \lceil \log c_e \rceil$. Clearly, the flows in $F(e)$ belong to F^M_i for $i \leq L(e)$. In other words, $F(e) \cap F^M_i = \emptyset$, for every $i > L(e)$.

Note that, in each class, by Property (P1), `FlowDec` assigns each color to at most one flow that uses e. This means that $D^t_a(k)$ has at most one flow that uses e from each set $C^t_a(z\tau + k)$, for $z = 0, \ldots, \lceil L(e)/\tau \rceil - 1$. Moreover, the size σ_i of a flow $f_i \in F^M_\ell$

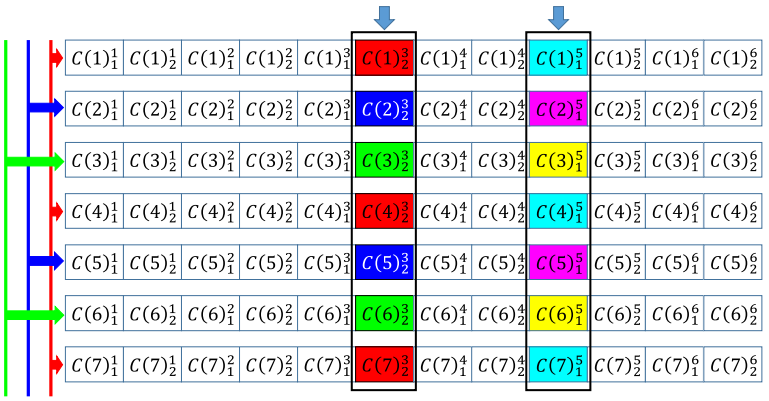

Fig. 2 In this figure, each row represents the output of `FlowDec` applied to a class. Initially, seven colors are used in each column, one per class. Next, `ColOptimize`, called with parameter $\tau = 3$, combines the colors across classes resulting in the use of 3 colors per column. For instance, in the sixth column colors Red, Blue, and Green are used. Note the non-standard notation in this figure; namely, $C(\ell)^i_a$ is used in place of $C^i_a(\ell)$ (Color figure online)

is bounded by $\alpha \cdot 2^\ell$. Hence, the combined size of the flows in $D^t_a(k)$ that go through e adds up to at most

$$\begin{aligned}\alpha \cdot c_e + 2^{L(e)-\tau} + 2^{L(e)-\tau-1} + \cdots &\le \alpha \cdot c_e + 2^{L(e)-\tau}(1 + 1/2 + 1/4 + \cdots) \\ &\le \alpha \cdot c_e + 2^{L(e)-\tau+1} \\ &\le \alpha \cdot c_e + 2^{2-\tau} c_e \le c_e,\end{aligned}$$

where the third inequality follows from $c_e > 2^{L(e)-1}$ and the last equality from $\tau = \log(1/(1-\alpha)) + 2$, equivalently $2^{2-\tau} = (1-\alpha)$. □

The main result of this section now directly follows from Lemma 3.

Theorem 3 *For any $\alpha \in (0, 1)$, there exists an offline $O(1+\log(\frac{1}{1-\alpha}))$ 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-that is instances where flows are not assumed to be α-small or α-large. Our treatment of large flows involves a reduction from Round-UFPP to the rectangle coloring problem (RCOL) which is discussed in Sect. 4.1. Next, in Sect. 4.2, we design an online algorithm for the RCOL instances that arise from the reduction. This algorithm is of independent interest. Later, in Sect. 4.3, we study `ProcLarges`, an online algorithm for Round-UFPP with $\frac{1}{4}$-large flows; this algorithm is based on the reduction to RCOL and the online algorithm we have designed for it. Finally, in Sect. 4.4, we

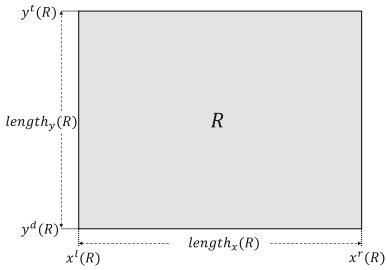

Fig. 3 A rectangle is specified by quadruple $(x^l(R), x^r(R), y^t(R), y^b(R))$

present our final algorithm for the general Round-UFPP instances which combines `ProcLarges` with an algorithm for $\frac{1}{4}$-small flows.

4.1 The Reduction from Round-UFPP with Large Flows to RCOL

Definition 7 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, $R = \{(x, y) \mid x^l(R) \leq x \leq x^r(R) \text{ and } y^b(R) \leq y \leq y^t(R)\}$ (see Fig. 3). 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 [11]. 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 4 [11] *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.*

A rectangle R_i in this scenario is called α-large or α-small if its corresponding flow f_i is α-large or α-small respectively. The main result here is that if all flows in F are $\frac{1}{k}$-large then an optimal coloring of $\mathcal{R}(F)$ is at most a factor of $2k$ worse than the optimal solution to the Round-UFPP instance arising from F. The following key lemma is crucial to the result.

Lemma 4 [11] *Let F be a feasible set of flows, and let $k \geq 2$ 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 1 *Let F be a feasible set of flows, and let $k \geq 2$ be an integer, such that every flow in F is $\frac{1}{k}$-large. Then, $\chi(\mathcal{R}(F)) \leq 2k\chi(F)$.*

Proof Consider an optimal coloring C of F with $\chi(F)$ colors. Apply Lemma 4 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 5 *Suppose there exists an offline α-approximation (online α-competitive) algorithm $\mathtt{A}$ 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 $\mathtt{A}$ to it. The solution is a valid Round-UFPP solution (Observation 4). Furthermore, by Corollary 1,

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

Importantly, 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 [13].

Definition 8 (*s-line-sparsity*) A collection of axis-parallel rectangles $\mathcal{R}$ is *s-line-sparse* if there exists a set of lines $L_{\mathcal{R}}$, called an *s-line-representative set* of $\mathcal{R}$, such that every rectangle $R \in \mathcal{R}$ is intersected by $1 \leq k_R \leq s$ lines in $L_{\mathcal{R}}$ (see Fig. 4 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 Sect. 4.2.1, we consider the 2-line-sparse case. Later, in Sect. 4.2.2, we study the general s-line-sparse case.

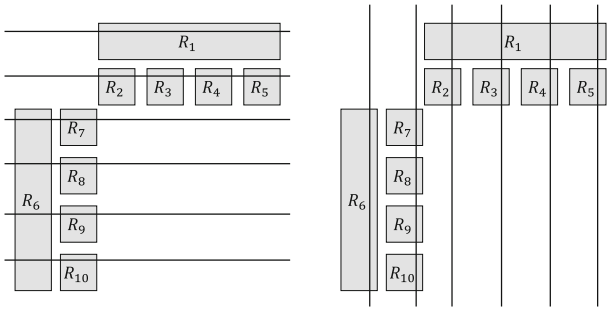

Fig. 4 A collection $\mathcal{R}$ of 4-line-sparse rectangles. The lines can be either **a** horizontal or **b** vertical

4.2.1 The 2-Line-Sparse Case

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

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

$$\mathcal{R}_l = \{R \in \mathcal{R} \mid T(R) = l \pmod 3\}, \text{ for } l = 0, 1, 2. \tag{10}$$

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

Lemma 6 *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 projections 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 $R \cap R' = \emptyset$.*

Proof (1) is easy to verify. Indeed, the projections of R and R' on the y-axis both contain $y(\ell_{T(R)})$; hence, their intersection is non-empty. Thus, R and R' intersect if and only if their projection on the x-axis has a non-empty intersection.

Next, we prove (2). Consider two rectangle $R, R' \in \mathcal{R}_l$, where $T(R) \neq T(R')$. Let $i = T(R)$ and $i' = T(R')$. Assume, without loss of generality, that $i < i'$. Note that $i' \geq i + 3$ by definition. Additionally, $y^t(R) < y(\ell_{i+1})$ since ℓ_i is the topmost line of L that intersects R. On the other hand, $y^b(R') > y(\ell_{i+1})$ since L is a 2-line-representative set of $\mathcal{R}$ meaning that at most two lines in L intersect R'. Consequently, the projection of R and R' on the y-axis have an empty intersection. Therefore, R and R' do not intersect. □

We will use the optimal 3-competitive online algorithm due to Kierstead and Trotter for ICP [22]. 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 (see Algorithm 5).

Algorithm 5: COL2SP

input : A rectangle $R \in \mathcal{R}$
input : The last state of COL2SP; a 2-representative-line set $L_{\mathcal{R}}$ for $\mathcal{R}$
output: A color for R

```
1 y ← T(R)%3;
2 return KT(R_y, R);
```

Basically, COL2SP computes a partition of $\mathcal{R}$ into $\mathcal{R}_0$, $\mathcal{R}_1$, and $\mathcal{R}_2$ as explained above and runs three instances of the KT algorithm in parallel one for each subset (see Fig. 5).

Lemma 7 *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 $9 \cdot \omega(\mathcal{R})$ colors*

Proof Let $Rec(\ell_i)$ denote the set of rectangles for which the line ℓ_i is the topmost line intersecting it. More precisely,

$$\mathrm{Rec}(\ell_i) = \{R \in \mathcal{R} \mid T(R) = i\}, \quad \text{for } i = 0, 1, \ldots, k.$$

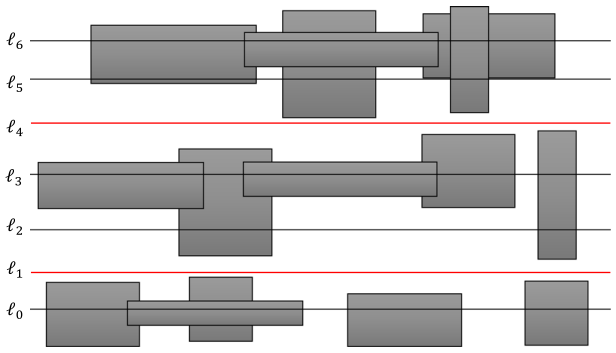

Fig. 5 An example for rectangle collection $\mathcal{R}_0$. The red lines are the ones whose index i satisfies $i = 1 \pmod 3$. None of the rectangles in $\mathcal{R}_0$ is intersected by a red line

Observe that, $\mathcal{R}_l$ defined in (10), satisfies

$$\mathcal{R}_l = \bigcup_{j=0}^{\lfloor (k-l)/3 \rfloor} \mathrm{Rec}(\ell_{3j+l}), \quad \text{for } l = 0, 1, 2.$$

Now, executing the KT algorithm on $\mathcal{R}_l$, is equivalent to executing the KT algorithm on $\mathrm{Rec}(\ell_l), \mathrm{Rec}(\ell_{3+l}), \mathrm{Rec}(\ell_{6+l}), \ldots$, simultaneously. Indeed, by Lemma 6, for every $R, R' \in \mathcal{R}_l$, we know that $R \cap R' = \emptyset$ if $R \in \mathrm{Rec}(\ell_i)$, $R' \in \mathrm{Rec}(\ell_j)$ and $i \neq j$. On the other hand, if $R, R' \in \mathrm{Rec}(\ell_i)$, part (2) of the lemma implies that the problem of coloring $\mathrm{Rec}(\ell_i)$ is the same as that of coloring intervals resulting from the projection of $\mathrm{Rec}(\ell_i)$ on the x-axis. Finally, since the KT algorithm is 3-competitive, COL2SP uses at most $3\omega(\mathrm{Rec}(\ell_i))$ colors to color $\mathrm{Rec}(\ell_i)$. Hence, overall, COL2SP colors $\mathcal{R}_l$ with at most

$$3 \cdot \max\{\omega(\mathcal{R}(\ell_i)) \mid i = l, 3+l, 2 \cdot 3 + l, \ldots, \lfloor (k-l)/3 \rfloor \cdot 3 + l\} \leq 3 \cdot \omega(\mathcal{R}_l) \leq 3 \cdot \omega(\mathcal{R})$$

colors for $l = 0, 1, 2$. However, since there could be rectangles $R_1 \in \mathcal{R}_i$ and $R_2 \in \mathcal{R}_j$, $i \neq j$, such that $R_1 \cap R_2 \neq \emptyset$, different colors must be used in each invocation. Consequently, the total number of colors used will be at most $9 \cdot \omega(\mathcal{R})$. □

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 set 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,

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

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

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

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 $L_{\mathcal{R}} = \{\ell_1, \ell_2, \ldots, \ell_k\}$ and define

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

Lemma 8 *For every $i \in \{0, \ldots, \lceil \log s \rceil\}$, $\mathrm{Lev}(i)$ is a 2-line-sparse set and $S(i)$ is a 2-line-representative set for* $\mathrm{Lev}(i)$.

Proof Fix an $i \in \{0, \ldots, \lceil \log s \rceil\}$. If $\text{Lev}(i) = \emptyset$, then it trivially is 2-line-sparse and any set of lines can serve as its 2-line-representative set. Now, suppose that $\text{Lev}(i) \neq \emptyset$ and pick an arbitrary rectangle $R \in \text{Lev}(i)$. We need to show that R intersects exactly either one or two lines in $S(i)$. By definition, we have that $2^i \leq \text{Deg}_{L_{\mathcal{R}}}(R) < 2^{i+1}$. On the other hand, $S(i) \subseteq L_{\mathcal{R}}$ contains one line for every 2^i lines of $L_{\mathcal{R}}$. Hence, R intersects at least one line and at most two lines in $\text{Lev}(i)$. □

We are ready to present an $O(\log s)$-competitive online algorithm, named `RectCol` (see Algorithm 6), for RCOL with a known line-representative set.

Algorithm 6: `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

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

Lemma 9 `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.*

Proof Consider an s-line-sparse set of rectangles $\mathcal{R}$ and an s-line-representative set L. By Lemma 8, $\text{Lev}(i)$ is 2-line-sparse and $S(i)$ is a 2-line-representative set of $\text{Lev}(i)$, for each $i = 0, \ldots, \lceil \log s \rceil$. Let $\#(\texttt{COL2SP}, \text{Lev}(i))$ denote the number of colors used by algorithm `COL2SP` to color $\text{Lev}(i)$. Observe that `RectCol` uses at most $\sum_{i=0}^{\lceil \log s \rceil} \#(\texttt{COL2SP}, \text{Lev}(i))$ colors. Furthermore, by Lemma 7, $\#(\texttt{COL2SP}, \text{Lev}(i)) \leq 9\omega(\mathcal{R})$, for every $i = 0, \ldots, \lceil \log s \rceil$. Therefore, Algorithm `RectCol` uses at most $9(\lceil \log s \rceil + 1)\omega(\mathcal{R})$ 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 $1/4 \leq \alpha \leq 1$. The online algorithm we have designed for RCOL needs 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 (Sect. 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 $L = \{l_0, l_1, \ldots, l_{s-1}\}$ 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 10 *L_1 is an s-line-representative set for $\mathcal{R}(\mathcal{F})$.*

Proof Since there are exactly s lines in L_1, every rectangle in $\mathcal{R}(F)$ is intersected by at most s lines. It remains to show that every rectangle is intersected by at least one line in L_1. To this end, consider an arbitrary rectangle $R_i \in \mathcal{R}(F)$. Since R_i corresponds to a $\frac{1}{4}$-large flow f_i, we have that $\sigma_i \geq b_i/4 = y_i^t/4$, where y_i^t is the top y-coordinate of the rectangle. Now, let j be an index such that

$$y(l_{j+1}) < y_i^t \leq y(l_j). \tag{11}$$

Note that such an index exists, since $y(l_0) = c_{\max}$ and $y(l_s) < c_{\min} = 1$. It follows from the right-hand side of (11) that $\frac{3}{4} y_i^t \leq \frac{3}{4} y(l_j)$. On the other hand, $y(l_{j+1}) = \frac{3}{4} y(l_j)$ by definition. Furthermore, $y_i^b < \frac{3}{4} y_i^t$ since $\sigma_i \geq y_i^t/4$. Therefore,

$$y_i^b \leq y(l_{j+1}) < y_i^t,$$

which implies that l_{j+1} intersects R_i. This completes the proof. □

The algorithm `ProcLarges`, for $\frac{1}{4}$-large flows, can be seen in Algorithm 7.

Algorithm 7: `ProcLarges`

input : A flow f
input : The last state of `ProcLarges`; a capacitated line graph $P = (V, E, c)$
output: A round for f

```
1 L ← ∅;
2 for i ← 0 to ⌈log_{4/3} c_max⌉ do
3 |   y(ℓ_i) ← (3/4)^i · c_max;
4 |   L ← L ∪ {ℓ_i};
5 Construct a rectangle R(f);
6 RectCol(R(f), L);
7 return the color index of R(f) ;
```

Theorem 5 `ProcLarges` *is an $O(\log \log c_{\max})$-competitive algorithm for instances of 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 9 states that Algorithm `RectCol` uses $O(\omega(\mathcal{R}(F)) \log \log c_{\max})$ colors. Lemma 5 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 (Algorithm 8) for Round-UFPP. `SolveRUFPP` simply uses procedure `ProcLarges` (Sect. 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 Sect. 3 or the 16-competitive algorithm in [16] in the offline case; and the 32-competitive algorithm in [17] in the online case.

Algorithm 8: `SolveRUFPP`

input : A flow f
input : The last state of `SolveRUFPP`; a capacitated line graph $P = (V, E, c)$
output: A round for f

1 **if** $\sigma_f \geq (1/4)b_f$ **then** `ProcLarges(f, P)` **else** `ProcSmalls(f, P)` **return**;

Theorem 6 *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 32-competitive [17]. On the other hand, by Theorem 5, `ProcLarges` is $O(\min(\log m, \log \log c_{\max}))$-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 Sect. 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 [17], 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.

Funding Funding was provided by National Science Foundation (Grant No. CCF-1422715) and Office of Naval Research.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Adamy, U., Erlebach, T.: Online coloring of intervals with bandwidth. In: Solis-Oba, R., Jansen, K. (eds.) Approximation and Online Algorithms, pp. 1–12. Springer, Berlin (2004)
2. Anagnostopoulos, A., Grandoni, F., Leonardi, S., Wiese, A.: A mazing 2+eps approximation for unsplittable flow on a path. CoRR arXiv:1211.2670 (2012)
3. Arkin, E.M., Silverberg, E.B.: Scheduling jobs with fixed start and end times. Discrete Appl. Math. **18**(1), 1–8 (1987)
4. Asplund, E., Grünbaum, B.: On a coloring problem. Math. Scand. **8**, 181–188 (1960)
5. Aumann, Y., Rabani, Y.: Improved bounds for all optical routing. In: Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '95, pp. 567–576. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (1995)
6. Azar, Y., Fiat, A., Levy, M., Narayanaswamy, N.: An improved algorithm for online coloring of intervals with bandwidth. Theor. Comput. Sci. **363**(1), 18–27 (2006)
7. Bansal, N., Chakrabarti, A., Epstein, A., Schieber, B.: A quasi-PTAS for unsplittable flow on line graphs. In: STOC'06, pp. 721–729 (2006)
8. Bansal, N., Friggstad, Z., Khandekar, R., Salavatipour, M.R.: A logarithmic approximation for unsplittable flow on line graphs. In: SODA'09, pp. 702–709 (2009)
9. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. J. ACM **48**(5), 1069–1090 (2001)
10. Bartlett, M., Frisch, A.M., Hamadi, Y., Miguel, I., Tarim, S.A., Unsworth, C.: The temporal knapsack problem and its solution. In: CPAIOR'05, pp. 34–48. Springer, Berlin (2005)
11. Bonsma, P., Schulz, J., Wiese, A.: A constant factor approximation algorithm for unsplittable flow on paths. In: FOCS'11, pp. 47–56 (2011)
12. Calinescu, G., Chakrabarti, A., Karloff, H., Rabani, Y.: An improved approximation algorithm for resource allocation. ACM Trans. Algorithms **7**(4), 48:1-48:7 (2011)
13. Chalermsook, P.: Coloring and maximum independent set of rectangles. In: APPROX. RANDOM 2011. Lecture Notes in Computer Science, vol. 6845. Springer, Berlin (2011)
14. Chekuri, C., Mydlarz, M., Shepherd, F.B.: Multicommodity demand flow in a tree. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. Lecture Notes in Computer Science, vol. 2719. Springer, Berlin
15. Darmann, A., Pferschy, U., Schauer, J.: Resource allocation with time intervals. Theoret. Comput. Sci. **411**(49), 4217–4234 (2010)
16. Elbassioni, K.M., Garg, N., Gupta, D., Kumar, A., Narula, V., Pal, A.: Approximation algorithms for the unsplittable flow problem on paths and trees. In: FSTTCS'12, pp. 267–275 (2012)
17. Epstein, L., Erlebach, T., Levin, A.: Online capacitated interval coloring. SIAM J. Discrete Math. **23**(2), 822–841 (2009)
18. Epstein, L., Levy, M.: Online interval coloring and variants. In: ICALP 2005. Lecture Notes in Computer Science, vol. 3580. Springer, Berlin
19. Garg, N., Vazirani, V.V., Yannakakis, M.: Primal-dual approximation algorithms for integral flow and multicut in trees. Algorithmica **18**(1), 3–20 (1997)
20. Golumbic, M.C., Lipshteyn, M., Stern, M.: Edge intersection graphs of single bend paths on a grid. Networks **54**(3), 130–138 (2009). https://doi.org/10.1002/net.20305
21. Kierstead, H.A.: The linearity of first-fit coloring of interval graphs. SIAM J. Discrete Math. **1**(4), 526–530 (1988)

22. Kierstead, H.A., Trotter, W.T.: An extremal problem in recursive combinatorics. Congr. Numer. **33**, 143–153 (1981)
23. Kostochka, A.: Coloring intersection graphs of geometric figures with a given clique number. In: Contemporary Mathematics, vol. 342. AMS (2004)
24. Kumar, S.R., Panigrahy, R., Russell, A., Sundaram, R.: A note on optical routing on trees. Inf. Process. Lett. **62**(6), 295–300 (1997)
25. Mihail, M., Kaklamanis, C., Rao, S.: Efficient access to optical bandwidth wavelength routing on directed fiber trees, rings, and trees of rings. In: Proceedings of IEEE 36th Annual Foundations of Computer Science, pp. 548–557 (1995)
26. Phillips, C.A., Uma, R.N., Wein, J.: Off-line admission control for general scheduling problems. J. Sched. **3**, 879–888 (2000)
27. Raghavan, P., Upfal, E.: Efficient routing in all-optical networks. In: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, STOC '94, pp. 134–143. ACM, New York, NY, USA (1994)
28. Zuckerman, D.: Linear degree extractors and the inapproximability of max clique and chromatic number. Theory Comput. **3**(6), 103–128 (2007)