A Mazing 2+ε Approximation for Unsplittable Flow on a Path*

ARIS ANAGNOSTOPOULOS, Sapienza University of Rome,
FABRIZIO GRANDONI, IDSIA, University of Lugano,
STEFANO LEONARDI, Sapienza University of Rome,
ANDREAS WIESE, Department of Industrial Engineering and Center for Mathematical Modeling, Universidad de Chile, Chile,

ACM Reference format:
Aris Anagnostopoulos, Fabrizio Grandoni, Stefano Leonardi, and Andreas Wiese. 2017. A Mazing 2+ε Approximation for Unsplittable Flow on a Path*. *ACM Trans. Algor.* 1, 1, Article 1 (January 2017), 22 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the *unsplittable flow on a path* problem (UFP) we are given a set of n tasks T and an undirected path $G = (V, E)$. Each edge e has a capacity $u_e \in \mathbb{N}^+$. Each task $i \in T$ is specified by a subpath $P(i)$ between the start (i.e. leftmost) vertex $s(i) \in V$ and the end (i.e. rightmost) vertex $t(i) \in V$, a demand $d(i) \in \mathbb{N}^+$, and a profit (or weight) $w(i) \geq 0$. For each edge $e \in E$, denote by T_e all tasks i using e, such that $e \in P(i)$. For any subset of tasks T', we define $w(T') := \sum_{i \in T'} w(i)$ and $d(T') := \sum_{i \in T'} d(i)$. The goal is to select a subset of tasks T' with maximum profit $w(T')$ such that $d(T' \cap T_e) \leq u_e$, for each edge e.

We can make the following simplifying assumptions w.l.o.g.: (i) For each task i, we assume that $d(i) \leq \min_{e \in P(i)}\{u_e\}$ (otherwise task i can be discarded); (ii) The edge capacities are all distinct (this can be achieved by slight perturbations and scaling, see [12]); (iii) Each node is either the start node or the end node of precisely one task, so that the number of nodes of the path is $2n$ (this can be enforced by duplicating and contracting edges in a proper way, similar to [6]).

UFP and variations of it are motivated by several applications in settings such as bandwidth allocation [10, 18, 25], caching [19], multicommodity demand flow [17], scheduling [4], or resource allocation [8, 13, 20, 26]. For example, edge capacities might model a given resource whose supply varies over a time horizon. Here, tasks correspond to jobs with given start and end times and each job has a fixed demand for the mentioned resource. The goal is then to

*An extended abstract of this paper appeared in the Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms (SODA 2014) [3].

1549-6325/2017/1-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

select the most profitable subset of jobs whose total demand at any time can be satisfied with the available resources.

When studying the problem algorithmically, a natural classification of the tasks is the following. For each task i define its *bottleneck capacity* as the smallest capacity $b(i) := \min\{u_e : e \in P(i)\}$ of any edge of $P(i)$. Let also the *bottleneck edge* $e(i)$ of i be the edge of i with capacity $b(i)$. For any value $\delta \in (0, 1]$ we say that a task i is δ-large if $d(i) \geq \delta \cdot b(i)$ and δ-*small* otherwise.

If all tasks are δ-small (δ-small instances), then the problem is well understood. As shown by Chekuri et al. [17, Corollary 3.4], for δ small enough, one can obtain an arbitrarily good approximation with LP-rounding techniques.

Theorem 1 ([17])**.** *For any $\delta \in (0, \frac{3-\sqrt{5}}{2})$, there is a $(1 + O(\sqrt{\delta}))$-approximation algorithm for δ-small instances of UFP.*

However, much remains unclear for the complementary case where all tasks are δ-large (δ-large instances), even if δ is very close to 1. A straightforward dynamic programming approach (where for each edge e one enumerates all possible subsets of tasks using e in an optimal solution) is doomed to fail: there are instances where in the optimal solution one edge of the path is used by *all* the tasks in the input. Also, for δ-large instances the canonical LP has an integrality gap of $\Omega(n)$ [14] (see Section 1.2 for the description of the canonical LP). The best known approximation factor proven by Bonsma et al. [12] for this setting is $2k$, where $k \in \mathbb{N}$ such that $\delta > \frac{1}{k}$ (and in particular $k \geq 2$). They reduce the problem to an instance of maximum independent set of rectangles and this approach inherently loses a factor of $2k \geq 4$ in the approximation ratio. In particular, their approximation ratio increases unboundedly if $\delta \to 0$. The best known $(6+\varepsilon)$-approximation algorithm for δ-large instances, for any $\delta > 0$, combines the approach above (with $k = 2$) with another algorithm, which is $2 + \varepsilon$ approximate for instances that are 1/2-small and δ-large at the same time. Combining this $(6 + \varepsilon)$-approximation with the result from Theorem 1, they obtain the currently best $(7 + \varepsilon)$-approximation algorithm for UFP [12] (for general instances).

1.1 Our Results and Techniques

In this paper, we present a polynomial-time approximation scheme (PTAS) for δ-large instances of UFP (for any constant $\delta > 0$), improving on the previously best $6 + \varepsilon$ approximation for the same case [12]. In combination with the algorithm from Theorem 1, our PTAS implies a $2 + \varepsilon$ approximation for arbitrary UFP instances, without any further assumptions (such as the common *no-bottleneck-assumption* $\max_i\{d(i)\} \leq \min_e\{u_e\}$ or restrictions on edge capacities). This improves on the previously best $7+\varepsilon$ approximation for the problem [12], and matches the best known approximation ratio for the much simpler case of uniform edge capacities [13]. We remark that, given our result, any further improvement of the approximation factor for UFP requires to consider δ-small and δ-large tasks at the same time.

Our PTAS for δ-large instances is based on a dynamic program (DP) and exploits the following geometrical viewpoint, inspired by [12] (but different in spirit and, as it turns out, more powerful). Let us represent edge capacities with a closed curve on the 2D plane (the *capacity curve*) as follows: We label nodes with integers from 1 to $2n$ going from left to right. For each edge $e = (v, v + 1)$, we draw a horizontal line segment (or segment for short) $[v, v+1] \times \{u_e\}$. Then we add a horizontal segment at the bottom, and vertical segments in a natural way to obtain a closed curve (see Figure 1). We represent each task i as a horizontal

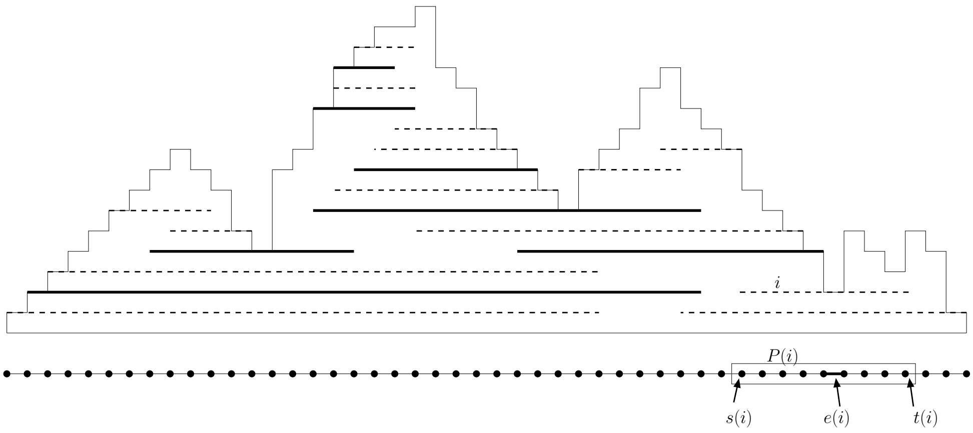


Fig. 1. An example of capacity curve and the maze, with segments associated to some tasks T' (dashed) and m-tasks M' (bold). Note that (T', M') is 2-thin.

segment $(s(i), t(i)) \times \{b(i)\}$. In particular, this segment lies underneath the capacity curve, and touches the horizontal segment corresponding to its bottleneck edge $e(i)$ (see Figure 1). (For comparison, our segments correspond to the upper sides of the rectangles from [12].) Note that, because all tasks are δ-large, the segment representation is sufficient to provide a rough estimate of task demands. At the same time, this notation turns out to be very convenient for the design of our DP and for its analysis.

The basic idea behind our approach is to sacrifice some tasks of the optimal solution, which we replace by profitless *maze tasks* (or *m-tasks* for short), that structure the area within the capacity curve into a *maze*. This maze has a tree topology: our DP *traverses* this tree from the leaves to the root, where the root is placed conventionally in the bottom-left corner of the capacity curve (it can be thought as the *exit* of the maze). We enforce the property that between any two consecutive (in a top–bottom ordering) m-tasks there are at most $k = O(1/\epsilon + 1/\delta)$ tasks above every edge e (see Figure 1). We will refer to this property as *k-thinness* later. This way, the DP is able to guess the tasks crossing any given corridor in an optimal solution.

One still remaining difficulty is that, when the DP computes a solution for some arising subproblem, we cannot afford to remember precisely which tasks were selected previously (this would result in too many DP-cells). To this end, the maze tasks have a second function: We use them to make it affordable to forget some decisions as we move from the leaves to the root. In particular, consider any edge e and let m be the m-task using e with highest bottleneck capacity $b(m)$. When we check the capacity constraint on edge e, we ignore all the *subcritical* tasks, which are tasks whose bottleneck capacity is below $\frac{\delta}{2}b(m)$. If the total capacity of all the tasks in the solution that use each edge e, ignoring the subcritical tasks, satisfies the capacity constraint of e, then we call the solution *weakly feasible*. We will show that a weakly feasible solution becomes feasible (in the usual sense) if we remove all maze tasks. The reason is that the capacity of the m-task with highest bottleneck capacity using edge e exceeds the total capacity of the subcritical tasks that use e, which we ignored in the definition of weak feasibility.

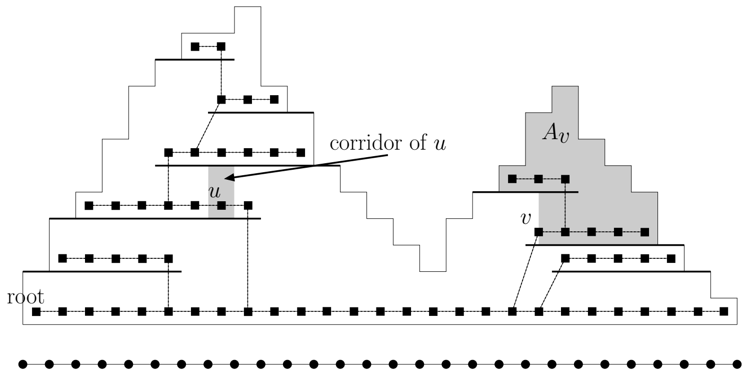


Fig. 2. An example of a maze and the corresponding tree topology, in which tree nodes are represented by squares.

Therefore, our DP computes a weakly feasible k-thin pair (T', M') where $T' \subseteq T$ is a set of tasks and M' is a set of m-tasks; in fact it computes the optimal weakly feasible solution among all pairs (T', M'), which is the solution that maximizes $w(T')$. The final output consists only of T', whose weight we seek to maximize. Because at the end we will remove the maze tasks of a computed solution we need to ensure that there is in fact a solution (T', M') where the weight of T' is almost the weight of the optimum T^*. We prove this by a nontrivial sequence of reductions where, eventually, the tasks of T^* are mapped into directed paths of a properly defined tree. On those paths we define a min-cost flow LP where each integral solution induces a k-thin pair (T', M') where the tasks T^* are (essentially) partitioned into T' and M'. The objective is to minimize $w(M')$. The claim then follows by showing that there exists a cheap fractional solution of weight at most $\varepsilon \cdot w(T^*)$, and that the LP matrix is totally unimodular.

The actual DP computing the optimal weakly feasible k-thin pair (T', M') is rather involved and technical. We next give an intuitive description, which highlights the key ideas. For the sake of simplicity, let us pretend that M' is given to us (whereas our DP will have to *guess* it along the way). Together with the capacity curve, the tasks in M' structure the plane into a maze. The maze induces a tree (see Figure 2), which guides our DP. In the maze tree, there is a node v for each *horizontal corridor* in the maze, which is characterized by an edge e and two *consecutive* m-tasks $m_\downarrow, m_\uparrow$ using e (at the top or the bottom of the capacity curve $m_\downarrow$ or $m_\uparrow$ may be undefined and then the corridor goes all the way to the bottom or to the top). We define the node on the bottom left corridor to be the root of the tree. For each node v we define an area A_v: draw a vertical segment connecting $m_\downarrow$ and $m_\uparrow$ above the midpoint of e. This segment partitions the maze into two disconnected regions. We define A_v to be the region not containing the bottom-left corner; see Figure 2. Then we place an edge (v, w) if $A_w \subset A_v$ and there is no other node u so that $A_w \subset A_u \subset A_v$. Due to the way we preprocessed the input instance, each node in the tree has at most two children: this helps to simplify the DP.

In the DP table there is a cell for each combination of a tree node v and a set of tasks $\bar{T}$. This set $\bar{T}$ contains a subset of *boundary* tasks i using the corridor of v (formally, $e \in P(i)$ and $b(m_\downarrow) < b(i) \leq b(m_\uparrow)$). Furthermore, $\bar{T}$ contains a subset of *critical* tasks for $m_\downarrow$ and $m_\uparrow$; those are tasks that share some edge with $P(m_\downarrow)$ (resp., $P(m_\uparrow)$) and whose bottleneck capacity is in the range $[\frac{\delta}{2}b(m_\downarrow), b(m_\downarrow))$ (resp., $[\frac{\delta}{2}b(m_\uparrow), b(m_\uparrow)))$. Observe that only those are the tasks *underneath* $m_\downarrow$ (resp., $m_\uparrow$) that are being considered when checking for weak feasibility—the rest are subcritial tasks, which, as we mentioned previously, are being ignored.

We show that, to obtain a good approximation, it is sufficient to consider subsets of at most $k = O(1/\varepsilon + 1/\delta)$ boundary tasks, and subsets of at most $O(1/\delta^2)$ critical tasks. This implies that the DP table has polynomial size. The value of each DP cell $(v, \bar{T})$ is the weight of the optimal solution to the following subproblem: select some tasks T' that lie completely in A_v such that $(\bar{T} \cup T', M')$ is weakly feasible and $w(T')$ is maximized. We prove that the value of a DP cell $(v, \bar{T})$ can be derived from the value of two DP cells $(v_1, \bar{T}_1)$, $(v_2, \bar{T}_2)$ associated to the children v_1, v_2 of v (or possibly only one cell, if v has only one child) that *are compatible* with $(v, \bar{T})$ and with each other. Hence, for computing the optimal solution of a cell $(v, \bar{T})$, we guess all possibilities for cells $(v_1, \bar{T}_1)$, $(v_2, \bar{T}_2)$, compute their optimal solutions, and select the best combination. The notion of *compatibility* then ensures locally that each such combination yields a feasible solution for the subproblem encoded in $(v, \bar{T})$. Leaf nodes form the base cases of the DP, and a suitable DP-cell associated to the root node gives the final solution.

1.2 Related Work

The best known polynomial-time approximation algorithm for UFP prior to this work achieved an approximation factor of $7 + \varepsilon$ [12]. This result improves on the previously best known polynomial time $O(\log n)$-approximation algorithm designed by Bansal et al. [7]. Bansal et al. [6] present a QPTAS for UFP assuming a quasi-polynomial bound on capacities and demands of the input instance. In terms of lower bounds, the problem is strongly NP-hard, even in the case of uniform edge capacities and unit profits [12, 19, 20].

Because of the difficulty of the general problem, researchers have studied special cases of UFP. The *resource allocation problem* (RAP) is the special case of UFP with uniform capacities. A 6-approximation for RAP is given by Phillips et al. [27]. The approximation ratio was later improved to 4 by Bar-Noy et al. [9], and finally to $2 + \varepsilon$ by Calinescu et al. [13]. The no-bottleneck assumption (NBA) requires that $\max_i\{d(i)\} \leq \min_e\{u_e\}$. Chekuri, Mydlarz, and Shepherd [17] give a $(2 + \varepsilon)$-approximation algorithm for UFP under the NBA. Observe that this generalizes the result in [13] since UFP instances with uniform capacities satisfy the NBA.

Several researchers addressed the problem of finding good LPs for UFP. The canonical LP for UFP has a variable $x_i \in [0, 1]$ for each task i and takes the following form:

$$\begin{aligned} \max \sum_i w(i)x_i & \\ \text{s.t.} \sum_{i: e \in P(i)} d(i)x_i \leq u_e & \qquad \forall e \\ x_i \in [0, 1]. & \end{aligned}$$

Unfortunately, this LP has an integrality gap of $\Omega(n)$ [14]. Adding further constraints, Chekuri, Ene, and Korula give an LP relaxation with an integrality gap of only $O(\log^2 n)$ [16], which was improved to $O(\log n)$ [15]. Anagnostopoulos et al. [2] describe a compact LP for the unweighted case of UFP with constant integrality gap. They also present an extended (i.e., containing extra variables besides the x_i's) compact LP for weighted UFP with constant integrality gap.

The problem of *unsplittable flow on a tree* (UFT) is the generalization of UFP where the input graph is a tree rather than a path. UFT is APX-hard, even for unit demands, edge capacities being either 1 or 2, and trees with depth 3 [21]. The currently best known approximation algorithm for UFT has a ratio of $O(\log^2 n)$ [16]. This was refinded and extended to a $O(k \log n)$-approximation by Adamaszek et al. [1] where k denotes the pathwidth of the given tree (always upper-bounded by $O(\log n)$) which works also if the objective function is given by a submodular function. Chekuri et al. [16] give a $O(\log(1/\gamma)/\gamma^3)$-approximation algorithm for UFT in the case that all tasks are $(1-\gamma)$-small. Under the NBA, Chekuri et al. [17] design a 48-approximation algorithm for UFT.

The *unsplittable flow* problem (UF) is a further generalization of UFP. Here the input graph $G = (V, E)$ is arbitrary, and the paths $P(i)$ are not specified in the input. A solution consists of a subset of selected tasks T', and a path $P(i)$ between $s(i)$ and $t(i)$ for each $i \in T'$. Azar and Regev [5] show that UF is NP-hard to approximate within a factor better than $O(|E|^{1-\varepsilon})$. They also present a $O(\sqrt{|E|})$ approximation under the NBA. This generalizes a similar approximation for the Edge Disjoint Path Problem (EDP) by Kleinberg [24].

1.3 Follow-up Results

After the publication of an extended abstract of this paper [3] several new results on UFP were found. Batra et al. [11] present a new QPTAS for the problem that does not require that the input data are quasi-polynomially bounded. In the same paper, the authors present a PTAS for the special case that the weight of each task is proportional to its demand, and a PTAS for a resource augmentation setting in which one is allowed to slightly shorten the path of each task while the compared optimum does not have this privilege. On a high level, the approach in that paper is similar as in our work: one proves that there is a near-optimal solution which is well-structured, similarly as the pairs (T', M') in our approach. Then one derives a dynamic program that computes a profitable solution with this property.

Extending this approach, Grandoni et al. [22] present PTASs for the special cases that there is an edge that is used by all tasks, that there are no two tasks whose paths are contained in each other, that one can select an arbitrary number of copies of each task, and that the profit of each task is proportional to the product of its demand and the length of its path, i.e., its area in a geometric sense. Very recently, Grandoni et al. [23] found a polynomial time $(31/16 + \epsilon)$-approximation algorithm for UFP.

2 DEFINITIONS AND METHODOLOGY

In this section we describe our methodology, which results in a polynomial-time $(1+\varepsilon)$-approximation algorithm for δ-large UFP instances (for any two given constants $\varepsilon, \delta > 0$). Recall that, in polynomial time, we can reduce the input instance so that each vertex is either the start or the end vertex of exactly one task in T (similarly to [6]). Thus, the number of nodes in the graph is $2n$.

Now we define the *maze tasks*, or m-tasks for short, which we use to structure our solution. For each pair of tasks i and j that share the same bottleneck edge e (possibly $i = j$), we define an m-task m with $P(m) = P(i) \cup P(j)$. Analogously to regular tasks, we set $b(m) = u_e$ and $e(m) = e$. Furthermore, we define $d(m) = \delta \cdot u_e$ and $w(m) = 0$. We remark that $d(i), d(j) \geq d(m)$. Let M_e be the m-tasks m with $e \in P(m)$. Note that because of the preprocessing described in Section 1, no two m-tasks with different bottleneck capacities share the same endpoint.

Our goal is to search for solutions in the form of *maze pairs* $(T', M') \in 2^T \times 2^M$, where we require for any two different m-tasks $m, m' \in M'$ that $b(m') \neq b(m'')$. Let $k = k(\varepsilon, \delta)$ be an integer constant to be defined later. We restrict our attention to maze pairs that are *k-thin* and *weakly feasible*, as defined below.

Intuitively, a maze pair (T', M') is k-thin if, for any edge e, between two consecutive segments (in a top–bottom ordering) associated to m-tasks from $M' \cap M_e$ there are at most k segments associated to tasks in $T' \cap T_e$ (see Figure 1).

Definition 2 (k-thinness)**.** *A maze pair (T', M') is k-thin if for every edge e and every set $T'' \subseteq T' \cap T_e$ with $|T''| > k$ there is an m-task $m \in M' \cap M_e$ such that $\min_{i \in T''}\{b(i)\} \leq b(m) < \max_{i \in T''}\{b(i)\}$.*

In Section 3 we prove that, for large enough k, there exists a k-thin maze pair $(\tilde{T}, \tilde{M})$ so that $\tilde{T}$ is a good approximation to the optimum T^* and $\tilde{T} \cup \tilde{M}$ is feasible (i.e., $d(\tilde{T} \cap T_e) + d(\tilde{M} \cap M_e) \leq u_e$ on each edge e).

Lemma 3. *For any $\varepsilon, \delta > 0$ there is a $k \in \mathbb{N}$ with $k = O(1/\varepsilon + 1/\delta)$, such that for any δ-large instance of UFP, there exists a k-thin maze pair $(\tilde{T}, \tilde{M})$ such that $w(\tilde{T}) \geq (1 - \varepsilon)w(T^*)$ and $\tilde{T} \cup \tilde{M}$ is feasible.*

However, we are not able to compute the most profitable k-thin maze pair in polynomial time. For this reason we relax the notion of feasibility of a maze pair (T', M') so that $T' \cup M'$ might not be feasible, but still T' alone is feasible (which is sufficient for our purposes). We need some definitions first. For every m-task $m \in M$ and any subset of tasks T', we partition the set $T'(m) := \{i \in T' : P(i) \cap P(m) \neq \emptyset\}$ of tasks of T' sharing some edge with m into three (disjoint) subsets:

- **(above tasks)** $abv(m, T') := \{i \in T'(m) : b(i) > b(m)\}$.
- **(critical tasks)** $crit(m, T') := \{i \in T'(m) : b(m) \geq b(i) \geq \frac{\delta}{2} b(m)\}$.
- **(subcritical tasks)** $subc(m, T') := \{i \in T'(m) : b(i) < \frac{\delta}{2} b(m)\}$.

We also define $abv_e(m, T') := abv(m, T') \cap T_e$, and we define analogously $crit_e(m, T')$ and $subc_e(m, T')$.

Definition 4 (Weak feasibility)**.** *A maze pair (T', M') is* weakly feasible *if for every edge e it holds that $d(abv_e(m_e, T')) + d(crit_e(m_e, T')) + d(m_e) \leq u_e$, where m_e is the m-task in $M' \cap M_e$ of largest bottleneck capacity, or $d(T' \cap T_e) \leq u_e$, if $M' \cap M_e = \emptyset$.*

Next we show that weak feasibility of a maze pair (T', M') implies feasibility of T'. The key argument is that for each edge e the task m_e compensates for the tasks in $subc_e(m_e, T')$ that were ignored in the definition of weak feasibility.

Lemma 5. *Let (T', M') be a weakly feasible maze pair. Then T' is feasible.*

Proof. Let $e_1, \ldots, e_m$ be the edges in nondecreasing order of capacity. We prove by induction on j that $d(T' \cap T_{e_j}) \leq u_{e_j}$ for all j. Consider first e_1. If there is no m-task using e_1, then the claim is true by definition. Otherwise let $m_1 = m_{e_1}$ be the (only) m-task in $M' \cap M_{e_1}$. All tasks $i \in T' \cap T_{e_1}$ must have $b(i) = u_{e_1}$ (in particular, they are critical for m_1). Thus $d(T' \cap T_e) \leq d(T' \cap T_e) + d(m_1) = d(abv_{e_1}(m_1, T')) + d(crit_{e_1}(m_1, T')) + d(m_1) \leq u_e$.

Now suppose by induction that there is a value $j \in \mathbb{N}$ such that $d(T' \cap T_{e_{j'}}) \leq u_{e_{j'}}$ for all $j' \in \{1, ..., j-1\}$. Consider the edge e_j. Once again if there is no m-task using e_j, then the claim is true by definition. Otherwise, let $m_j := m_{e_j}$. Consider the subcritical tasks $SC := subc_{e_j}(m_j, T')$. By definition, e_j is not the bottleneck edge of any task in SC. We partition SC into the sets SC_L and SC_R, containing the tasks with bottleneck edge on the left of e_j and on the right of e_j, respectively. Consider the set SC_L. Let $i_L \in SC_L$ be a task with maximum bottleneck capacity in SC_L and let e_L be its bottleneck edge. By the definition of SC_L and i_L, all tasks in SC_L use e_L and $u_{e_L} = b(i_L) < \frac{\delta}{2} \cdot b(m_j)$. Using the induction hypothesis on e_L, we obtain that $d(SC_L) = d(SC_L \cap T_{e_L}) \le d(T' \cap T_{e_L}) \le u_{e_L} < \frac{\delta}{2} \cdot b(m_j)$. Similarly, we obtain that $d(SC_R) < \frac{\delta}{2} \cdot b(m_j)$. Since $d(m_j) = \delta \cdot b(m_j)$ the m-task m compensates for all tasks in SC, that is,

$$\begin{aligned} d(subc_{e_j}(m_j, T')) = d(SC) &= d(SC_L) + d(SC_R) \\ &< \delta \cdot b(m_j) = d(m_j). \end{aligned}$$

Hence

$$\begin{aligned} d(T' \cap T_{e_j}) &= d(abv_{e_j}(m_j, T')) + d(crit_{e_j}(m_j, T')) \\ &\quad + d(subc_{e_j}(m_j, T')) \\ &\le d(abv_{e_j}(m_j, T')) + d(crit_{e_j}(m_j, T')) \\ &\quad + d(m_j) \\ &\le u_{e_j}, \end{aligned}$$

where the last inequality follows from the weak feasibility of (T', M'). □

Note that the maze pair $(\tilde{T}, \tilde{M})$ obtained in Lemma 3 is feasible so, by definition, it is also weakly feasible. In Section 4 we present a polynomial-time dynamic program that computes the weakly feasible k-thin maze pair with highest profit.

Lemma 6. *For any constants $\delta \in (0, 1]$ and $k \in \mathbb{N}^+$, there is a dynamic program with running time $n^{O(k+1/\delta^2)}$ that computes a weakly feasible k-thin maze pair (T', M') of largest profit $w(T')$ for δ-large instances of UFP.*

A crucial property that we exploit in the design of our dynamic program is that for each m-task in a weakly feasible maze pair the number of critical tasks is bounded by a constant depending only on δ.

Lemma 7. *Let (T', M') be a weakly feasible maze pair and $m \in M'$. It holds that $|crit(m, T')| \le ncrit(\delta) := \frac{4}{\delta^2} + \frac{1}{\delta}$.*

PROOF. First recall that, by Lemma 5, T' is a feasible solution. Consider the tasks $i \in crit(m, T')$ with $b(i) = b(m)$. Because all tasks are δ-large, there can be at most $1/\delta$ such tasks. The remaining tasks $i \in crit(m, T')$ have $b(i) < b(m)$ and must use the leftmost edge e_L of $P(m)$ or the rightmost edge e_R of $P(m)$ (or both). Consider the tasks C_L of the first type: we will show that $|C_L| \le 2/\delta^2$. A symmetric argument holds for the remaining tasks C_R, hence giving the claim. Consider the task $i_L \in C_L$ that has the largest $b(i_L)$. By the definition of C_L and i_L all tasks in C_L must use $e(i_L)$ and $b(i_L) \le b(m)$. Each task $i \in C_L$ is critical for m and thus $b(i) \ge \frac{\delta}{2} b(m)$. Also, i is δ-large and so $d(i) \ge \delta b(i) \ge \frac{\delta^2}{2} b(m)$. Therefore, there can be at most $b(i_L)/(\frac{\delta^2}{2} b(m)) \le \frac{2}{\delta^2}$ such tasks. □

By combining Lemmas 3, 5 and 6 we obtain the main theorem of this paper.

Theorem 8. *For any constant $\delta > 0$, there is a PTAS for δ-large instances of UFP.*

Combining Theorem 8 with Theorem 1, we obtain the following corollary.

Corollary 9. *For any constant $\varepsilon > 0$, there is a polynomial-time $(2 + \varepsilon)$ approximation algorithm for UFP.*

3 A THIN PROFITABLE MAZE PAIR

In this section we prove Lemma 3, that is, we show that for any instance there is a k-thin maze pair $(\tilde{T}, \tilde{M})$ such that $w(\tilde{T}) \geq (1-\varepsilon)OPT$ and $\tilde{T} \cup \tilde{M}$ is feasible. Recall that, by assumption, the vertices of the graph are labeled by $1, \dots, 2n$ from left to right. Let $\ell(i)$ denote the segment $(s(i), t(i)) \times \{b(i)\}$ associated to each task $i \in T^*$. Define $L := \{\ell(i) : i \in T^*\}$ and $w(\ell(i)) := w(i)$. We say that a segment $(a, b) \times \{y\}$ *contains an edge* $e = (v, v+1)$ if $(v, v+1) \subseteq (a, b)$.

We want to select a subset $L' \subseteq L$ such that $w(L') := \sum_{\ell(i) \in L'} w(\ell(i))$ is at most $\varepsilon \cdot w(T^*)$ and any vertical segment $\{x\} \times (y_b, y_t)$ intersecting more than k segments in L intersects at least one segment in L'. We call a set L' with the latter property *k-thin for L*. As we will show, for proving Lemma 3 it suffices to find a k-thin set L' for L because of the following transformation of L' into a maze-pair $(T(L'), M(L'))$. We define $T(L') := \{i : \ell(i) \in L \setminus L'\}$. For constructing $M(L')$ we group the segments in L' according to the bottleneck edges of their corresponding tasks. For each edge e, we define $L'_e := \{\ell(i) \in L' : e(i) = e\}$. Now for each edge e with $L'_e \neq \emptyset$ we add to $M(L')$ the m-task $m_e \in M$ with path $P(m) = \cup_{\ell(i) \in L'_e} P(i)$. Note that $P(m) = P(i_L) \cup P(i_R)$ for the task $i_L \in L'_e$ with leftmost start vertex and the task $i_R \in L'_e$ with rightmost end vertex (in a sense, we glue i_L and i_R together to form an m-task). Hence m_e is a well-defined m-task. Observe that, as required in the definition of a maze pair, we have $b(m') \neq b(m'')$ for any two distinct $m', m'' \in M(L')$.

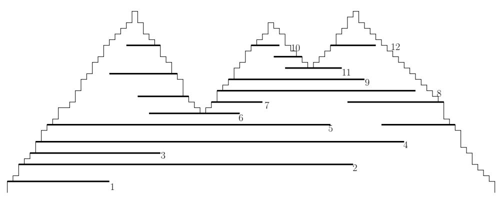
Lemma 10. *If a set $L' \subseteq L$ is k-thin for L, then the maze pair $(T(L'), M(L'))$ is $(k+1/\delta)$-thin and $T(L') \cup M(L')$ is feasible.*

PROOF. The proof is based on the similarity of the definitions of k-thinness for segments and for maze pairs. Some extra work is required because some line segments may share the same bottleneck edge and then overlap. Consider any edge $e = (u, u+1)$, and any set of $k + 1/\delta + 1$ tasks $T' \subseteq T^* \cap T_e$. We define $\{i_1, i_2, \dots, i_{k+1}\} \subseteq T'$ to be $k+1$ of them with lowest bottleneck capacity, in nondecreasing order of bottleneck capacity. Let $b_{max} := \max_{i \in T'}\{b(i)\}$. Since T^* is feasible, and since the tasks in T^* are δ-large, there cannot be more than $1/\delta$ tasks in T' of bottleneck capacity equal to b_{max}. It follows that $b(i_j) < b_{max}$ for all $1 \leq j \leq k+1$.

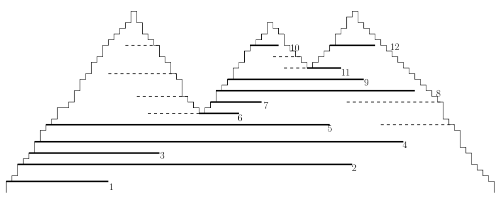
Consider a vertical segment ℓ' with x-coordinate $u + \frac{1}{2}$ that intersects $\ell(i_1), \dots, \ell(i_{k+1})$. Since L' is k-thin, ℓ' must intersect some segment $\ell(i^*) \in L'$. Segment $\ell(i^*)$ corresponds to a task i^*; in turn, to this task corresponds an m-task $m \in M(L')$ with $b(i_1) \leq b(m) = b(i^*) \leq b(i_{k+1}) < b_{max}$. Hence $(T(L'), M(L'))$ is $(k + 1/\delta)$-thin.

To show the feasibility of $T(L') \cup M(L')$ recall that T^* is feasible and all tasks in T^* are δ-large. Furthermore, each m-task has capacity $\delta \cdot b(m)$. Therefore, on every edge e each m-task m_e uses at most as much capacity as the tasks from T^* whose segments are in L'_e (the latter tasks in a sense were replaced by m_e). $\square$

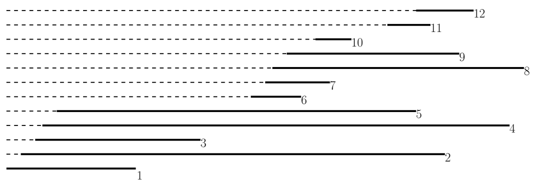
Next we reduce the problem of finding a k-thin set L' with low weight to the case that each segment $\ell(i)$ starts at $e(i)$ and either goes only to the right or only to the left. See Figures 3(a) and 3(b). Formally, we split each segment $\ell(i)$ into two segments $\ell_L(i)$ and $\ell_R(i)$ such that $\ell_L(i)$ contains the edges of $P(i)$ between $s(i)$ and the right vertex of the bottleneck edge $e(i)$ and symmetrically for $\ell_R(i)$. So $\ell_L(i)$ and $\ell_R(i)$ overlap on $e(i)$. We set $w(\ell_L(i)) = w(\ell_R(i)) = w(i)$. We define $L_L := \{\ell_L(i) : \ell(i) \in L\}$ and $L_R := \{\ell_R(i) : \ell(i) \in L\}$. The next lemma shows that it suffices to find low weight k-thin sets for L_L and L_R.



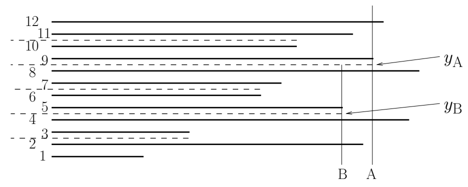
(a) Initial instance. The numbers identify some of the tasks.



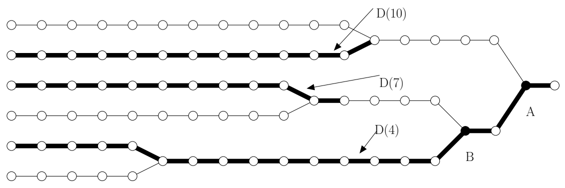
(b) Splitting the segments to the ones in L_R (in bold) and to L_L (dashed).



(c) Line segments in L_R shifted up (distances distorted) and extended to the left (in dashed).



(d) Decomposition of the segments for $k = 4$. The dashed lines indicates the values (y_0 in the text) where the respective interval I_w is split.



(e) The tree created from the decomposition of (d) (for representation issues, arc directions (all towards the left) are omitted, and some nodes of degree 2 are contracted). A and B indicate the nodes corresponding to the splittings in (d).

Fig. 3. Construction of the maze

Lemma 11. *Given k-thin sets L'_L for L_L and L'_R for L_R, there is a $2k$-thin set L' for L with $w(L') \leq w(L'_L) + w(L'_R)$.*

PROOF. We add a segment $\ell(i)$ to L' if and only if $\ell_L(i) \in L'_L$ or $\ell_R(i) \in L'_R$. It follows directly that $w(L') \leq w(L'_L) + w(L'_R)$. Now any vertical segment ℓ' crossing at least $2k+1$ segments in L' must either cross $k+1$ segments from L_L or $k+1$ segments from L_R. Thus, ℓ' crosses a segment in L'_L or a segment in L'_R, and hence ℓ' crosses a segment in L'. $\square$

Consider now only the segments L_R (a symmetric argument holds for L_L). The next step is to reduce the problem to the case where, intuitively speaking, the edge capacities are strictly increasing and all segments contain the leftmost edge of the graph (we shift up segments and then extend them to the left). To simplify the description of the next step, we also enforce that new segments have different y-coordinates. Formally, let us assume that task labels i are integers between 1 and n (in any order). For each $\ell_R(i) = (v, u) \times \{b(i)\} \in L_R$, we construct a segment $(1, u) \times \{b(i) + M \cdot v + \varepsilon \cdot i\}$, which we denote by $\tilde{\ell}_R(i)$. Here $M := 1 + \max_e\{u_e\}$

and $\varepsilon = \frac{1}{n+1}$ (so that $\varepsilon \cdot i < 1$). Define $\tilde{L}_R := \{\tilde{\ell}_R(i) : i \in T^*\}$ and $w(\tilde{\ell}_R(i)) = w(i)$. (See Figure 3(c).)

Lemma 12. *Given a k-thin set $\tilde{L}'_R$ for $\tilde{L}_R$, there is a k-thin set L'_R for L_R with $w(\tilde{L}'_R) = w(L'_R)$. A symmetric claim holds for $\tilde{L}_L$ and L_L.*

Proof. We prove the first claim only, the proof of the second one being symmetric. Let $L'_R := \{\ell_R(i) \in L_R : \tilde{\ell}_R(i) \in \tilde{L}'_R\}$. Clearly $w(L'_R) = w(\tilde{L}'_R)$. Consider any vertical segment $\{x\} \times (y_b, y_t)$ that intersects at least $k+1$ segments from L_R. Let $\ell_R(i_1), \ldots \ell_R(i_{k+1})$ be $k+1$ such segments of lowest capacity, breaking ties according to the lowest label i of the corresponding tasks. To prove the lemma it suffices to show that at least one such segment $\ell_R(i_{j^*})$ belongs to L'_R.

W.l.o.g., assume that for any $1 \leq j \leq k$, $s(i_j)$ is equal to or to the left of $s(i_{j+1})$, and $i_j < i_{j+1}$ if $s(i_j) = s(i_{j+1})$. Then by construction $\tilde{y}_1 < \ldots < \tilde{y}_{k+1}$, where $\tilde{y}_j$ is the y-coordinate of segment $\tilde{\ell}_R(i_j)$. Consider a vertical segment $\{x\} \times (\tilde{y}_1 - \varepsilon', \tilde{y}_{k+1} + \varepsilon')$. For $\varepsilon' > 0$ small enough, we can assume that ℓ' intersects precisely the segments $\tilde{\ell}_R(i_1), \ldots, \tilde{\ell}_R(i_{k+1})$. Hence $\tilde{\ell}_R(i_{j^*}) \in \tilde{L}'_R$ for some $1 \leq j^* \leq k+1$. It follows from the definition of L'_R that $\ell_R(i_{j^*}) \in L'_R$ as required. □

It remains to prove that there is a k-thin set for $\tilde{L}_R$ whose weight is bounded by $O(1/k)w(\tilde{L}_R)$. We do this by reducing this problem to a min-cost flow problem in a directed tree network. Assume w.l.o.g. that $k \in \mathbb{N}$ is even (this assumption has the only purpose to avoid ceilings and floors). We consider the following hierarchical decomposition of the segments in $\tilde{L}_R$, which corresponds to a (directed) rooted out-tree $\mathcal{D}$ (see Figures 3(d) and 3(e)). We construct $\mathcal{D}$ iteratively, starting from the root. Each node w of $\mathcal{D}$ is labelled with a triple (e_w, I_w, R_w), where e_w is an edge in E, $I_w \subseteq [0, \infty)$ is an interval, and R_w contains all segments that contain e and whose y-coordinate is in I_w (the *representative segments* of w). Let $e_r \in E$ be the rightmost edge that is contained in at least $k-1$ segments. We let the root r of $\mathcal{D}$ be labelled with $(e_r, [0, \infty), R_r)$. For any constructed node w, if e_w is the leftmost edge of the graph, then w is a leaf. Otherwise, consider the edge e' to the left of e_w, and let R' be the segments in I_w that contain e'. Note that, by the initial preprocessing of the instance, each edge can be the rightmost edge of at most one segment (task), hence $|R'| \leq |R_w| + 1$. If $|R'| < k$, we append to w a child w' (with a directed arc (w, w')) with label (e', I_w, R'). Otherwise (i.e., if $|R'| = k$), we append to w two children w_b and w_t, which are labelled as follows. Let $\tilde{\ell}_R(i_1), ..., \tilde{\ell}_R(i_k)$ be the segments in R', sorted increasingly by y-coordinate (here we exploit the fact that y-coordinates are all distinct). We partition R' into $R_b = \{\tilde{\ell}_R(i_1), ..., \tilde{\ell}_R(i_{k/2})\}$ and $R_t = \{\tilde{\ell}_R(i_{k/2+1}), ..., \tilde{\ell}_R(i_k)\}$. Let y_0 be a value such that all segments in R_b have a y-coordinate strictly smaller than y_0 and all segments in R_t have a y-coordinate strictly greater than y_0. We label w_b and w_t with $(e', I_w \cap [0, y_0), R_b)$ and $(e', I_w \cap [y_0, \infty), R_t)$, respectively.

Consider a given segment $\tilde{\ell}_R(i) \in \tilde{L}_R$, and the nodes w of $\mathcal{D}$ that have $\tilde{\ell}_R(i)$ as one of their representative segments R_w. Then the latter nodes induce a directed path $\mathcal{D}(i)$ in $\mathcal{D}$. To see this, observe that if $\tilde{\ell}_R(i) \in R_w$, then either w is a leaf or $\tilde{\ell}_R(i) \in R_{w'}$ for exactly one child w' of w. Furthermore, each $\tilde{\ell}_R(i)$ belongs to R_w for some leaf w of $\mathcal{D}$ (i.e., no $\mathcal{D}(i)$ is empty).

We call a set of segments $\tilde{L}'_R \subseteq \tilde{L}_R$ a *segment cover* if for each node w of $\mathcal{D}$ it holds that $R_w \cap \tilde{L}'_R \neq \emptyset$.

Lemma 13. *If $\tilde{L}'_R \subseteq \tilde{L}_R$ is a segment cover, then $\tilde{L}'_R$ is $2k$-thin for $\tilde{L}_R$. A symmetric claim holds for $\tilde{L}_L$.*

Proof. We prove the first claim only, the proof of the second one being symmetric. Consider any vertical segment $\ell' = \{x\} \times (y_b, y_t)$ crossing at least $2k+1$ segments from $\tilde{L}_R$, and let $\tilde{L}''$ be $2k+1$ such segments of lowest y-coordinate. Let also $e = (u, u+1)$ be the edge such that $x \in (u, u+1)$, and $\tilde{\ell}_R(i_1), \ldots, \tilde{\ell}_R(i_h)$ be the segments containing edge e in increasing order of y coordinate. Observe that segments $\tilde{L}''$ induce a subsequence $\tilde{\ell}_R(i_j), \tilde{\ell}_R(i_{j+1}), \ldots, \tilde{\ell}_R(i_{j+2k})$ of $\tilde{\ell}_R(i_1), \ldots, \tilde{\ell}_R(i_h)$. Furthermore, the representative sets R_w of nodes w such that $e_w = e$ partition $\tilde{\ell}_R(i_1), \ldots, \tilde{\ell}_R(i_h)$ into subsequences, each one containing between $k/2$ and $k-1$ segments. It follows that there must be one node w' such that $R_{w'} \subseteq \{\tilde{\ell}_R(i_j), \ldots, \tilde{\ell}_R(i_{j+2k})\}$. Since $\tilde{L}'_R \cap R_{w'} \neq \emptyset$ by assumption, it follows that $\tilde{\ell}_R(i_{j^*}) \in \tilde{L}'_R$ for some $j \leq j^* \leq j+2k$. ☐

It remains to show that there is a segment cover with small weight.

Lemma 14. *There exists a segment cover $\tilde{L}'_R \subseteq \tilde{L}_R$ with $w(\tilde{L}'_R) \leq \frac{2}{k} \cdot w(\tilde{L}_R)$ (where k is the parameter used in the construction of $\mathcal{D}$).*

Proof. We can formulate the problem of finding a $\tilde{L}'_R$ satisfying the claim as a flow problem. We augment $\mathcal{D}$ by appending a dummy node w' to each leaf node w with a directed edge (w, w') (so that all the original nodes are internal) and extend the paths $\mathcal{D}(i)$ consequently (so that each path contains exactly one new edge (w, w')). We define a min-cost flow problem, specified by a linear program. For each directed path $\mathcal{D}(i)$ we define a variable $x_i \in [0,1]$. Let A denote the set of all arcs in $\mathcal{D}$. For each arc a denote by T_a all values i such that $\mathcal{D}(i)$ uses a. For arguing about the flow problem, we consider the following linear program:

$$\min \sum_{i:\ell(i)\in\tilde{L}_R} w(i)\cdot x_i$$
$$\text{s.t.} \sum_{i\in T_a} x_i \geq 1 \qquad \forall a \in A$$
$$x_i \geq 0 \qquad \forall \ell(i) \in \tilde{L}_R.$$

By the construction of $\mathcal{D}$ every arc is used by at least $k/2$ paths. Hence, the linear program has a fractional solution of weight $\sum_i w(i) \cdot \frac{2}{k} = \frac{2}{k} \cdot w(\tilde{L}_R)$, which is obtained by setting $x_i := 2/k$ for each i. Since the underlying network $\mathcal{D}$ is a directed tree and all paths follow the direction of the arcs, the resulting network flow matrix is totally unimodular, see [28]. Therefore, there exists also an integral solution with at most the same weight. This integral solution induces the set $\tilde{L}'_R$. ☐

Now the proof of Lemma 3 follows easily from the previous reductions.

Proof of Lemma 3. Suppose we are given the optimal solution T^*. As described above, we construct the sets L, L_L, L_R, $\tilde{L}_L$, and $\tilde{L}_R$. We compute segment covers $\tilde{L}'_L$ for $\tilde{L}_L$ and $\tilde{L}'_R$ for $\tilde{L}_R$ as described in the proof of Lemma 14. By Lemma 13 they are $2k$-thin for $\tilde{L}_L$ and $\tilde{L}_R$, respectively. By Lemma 12 we obtain $2k$-thin sets L'_L and L'_R for L_L and L_R, respectively, with $w(L'_L) = w(\tilde{L}'_L)$ and $w(L'_R) = w(\tilde{L}'_R)$. By Lemma 11 this yields a $4k$-thin set L' for L whose weight is bounded by $w(L'_L) + w(L'_R)$. Finally, set $(\tilde{T}, \tilde{M}) := (T(L'), M(L'))$. This maze pair is feasible by definition. Furthermore, by Lemma 10, it is $(4k + \frac{1}{\delta})$-thin and its weight is bounded by $w(\tilde{T}) \leq w(\tilde{L}'_L) + w(\tilde{L}'_R) \leq \frac{2}{k} \cdot (w(\tilde{L}_L) + w(\tilde{L}_R)) \leq \frac{4}{k} \cdot w(L) = \frac{4}{k} \cdot w(T^*)$. By setting the parameter k in the construction of $\mathcal{D}$ to be $4/\varepsilon$ we obtain an $O(1/\varepsilon + 1/\delta)$-thin maze pair, completing the proof of the lemma.

4 THE DYNAMIC PROGRAM

In this section we present a DP that computes the weakly feasible k-thin maze pair $(\tilde{T}, \tilde{M})$ with maximum weight $w(\tilde{T})$, and thus prove Lemma 6. Let $k = O(1/\varepsilon + 1/\delta)$ (as suggested by Lemma 3). To simplify the description and analysis of our DP, we introduce the following assumptions and notations. For technical reasons, we add edges e_L and e_R to the left and right of the input graph, respectively, and set their capacity to zero (those edges are used by no task). For notational convenience, we also add to M two special dummy m-tasks $\bot$ and $\top$. The paths of $\bot$ and $\top$ span all the edges of the graph, and they both have demand zero. Furthermore, $b(\top) := +\infty$ and $b(\bot) := 0$. In particular, with these definitions we have that $abv_e(\top, T') = crit_e(\top, T') = \emptyset$, $abv_e(\bot, T') = T' \cap T_e$, and $crit_e(\bot, T') = \emptyset$. We let $e(\bot) = e_R$, and we leave $e(\top)$ unspecified. However, when talking about weak-feasibility and k-thinness of a maze pair (T', M') we will ignore the dummy m-tasks, that is, we will implicitly consider $(T', M' \setminus \{\bot, \top\})$.

For any $e \in E$, $T' \subseteq T$, and any two m-tasks m' and m'' with $b(m') < b(m'')$, the *boundary tasks* in T' for the triple (e, m', m'') are the tasks

$$bound_e(m', m'', T') := \{i \in T' \cap T_e : b(m') < b(i) \leq b(m'')\}.$$

Intuitively, boundary tasks i are the tasks using edge e such that the segment corresponding to i is sandwiched between the segments corresponding to m' and m''.

In our DP table we introduce a cell for each tuple of the form $c = (e, m_\uparrow, C_\uparrow, m_\downarrow, C_\downarrow, B)$ where:

- e is an edge;
- $m_\downarrow \in M_e$ and $m_\uparrow \in M_e$, $b(m_\downarrow) < b(m_\uparrow)$;
- $C_\downarrow \subseteq crit(m_\downarrow, T)$ and $C_\uparrow \subseteq crit(m_\uparrow, T)$, with $|C_\uparrow|, |C_\downarrow| \leq ncrit(\delta)$;
- $B \subseteq bound_e(m_\downarrow, m_\uparrow, T)$, with $|B| \leq k$.

Recall that $ncrit(\delta) = \frac{4}{\delta^2} + \frac{1}{\delta}$ (see Lemma 7). Intuitively, $C_\downarrow$ (resp., $C_\uparrow$) are the critical tasks associated to $m_\downarrow$ (resp., $m_\uparrow$). Observe that $C_\downarrow$ and B are disjoint, whereas $C_\uparrow$ might overlap with both $C_\downarrow$ and B. For such a cell to exist we further impose the following *consistency property*: we require that $(B \cup C_\downarrow \cup C_\uparrow, \{m_\downarrow, m_\uparrow\})$ is weakly feasible and that for $T' = B \cup C_\downarrow \cup C_\uparrow$, one has $crit(m_\downarrow, T') = C_\downarrow$, $crit(m_\uparrow, T') = C_\uparrow$, and $bound_e(m_\downarrow, m_\uparrow, T') = B$.

Given a DP cell $c = (e, m_\uparrow, C_\uparrow, m_\downarrow, C_\downarrow, B)$, as a shorthand notation we use $e(c) := e$, $m_\uparrow(c) := m_\uparrow$ and similarly for the other indices of the cell. We also define $e_\downarrow = e_\downarrow(c) := e(m_\downarrow)$ and $e_\uparrow = e_\uparrow(c) := e(m_\uparrow)$ (we set $e_\uparrow = e$ if $m_\uparrow = \top$). The idea behind a cell c is as follows. We define $E(c)$ as the set of edges between $e_\uparrow$ (included) and $e_\downarrow$ (excluded) (if $e_\uparrow = e_\downarrow$, we assume $E(c) = \emptyset$). We define $T(c)$ as the set of tasks i with bottleneck edge in $E(c)$ such that $b(i) > b(m_\uparrow)$ or $P(i)$ does not contain e. We define $M(c)$ similarly w.r.t. m-tasks. For a geometric intuition we can think of cell c as defining an area such that $T(c)$ and $M(c)$ lie entirely inside—see Figure 4.

Our goal is to compute the maze-pair (T_c, M_c) with $T_c \subseteq T(c)$ and $M_c \subseteq M(c)$ with maximum weight $w(c) := w(T_c)$ such that:

(1) $(T_c \cup B \cup C_\downarrow \cup C_\uparrow, M_c \cup \{m_\downarrow, m_\uparrow\})$ is weakly feasible;
(2) $(T_c \cup B, M_c \cup \{m_\downarrow, m_\uparrow\})$ is k-thin;
(3) If $i \in crit(m_\uparrow, T_c)$ then $i \in C_\uparrow$ (*inclusion property*).

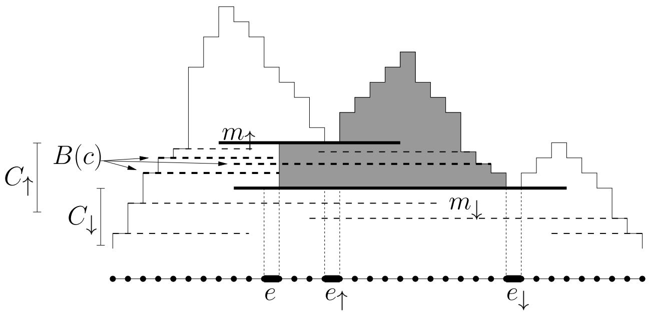


Fig. 4. Tasks $B \cup C_\downarrow \cup C_\uparrow$ (dashed) and area associated to a DP cell c. Tasks in $C_\uparrow$ ($C_\downarrow$) use a common edge with $m_\uparrow$ ($m_\downarrow$). Tasks (resp., m-tasks) that lie entirely within the shaded area are those that belong to $T(c)$ (resp., $M(c)$).

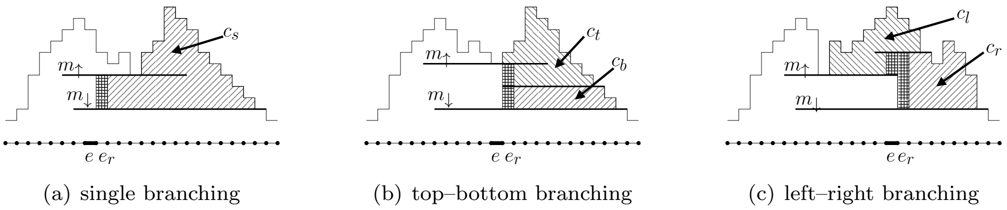


Fig. 5. The three branching cases. The area of the cell c is the area of the subcell(s) c_s, (c_t, c_b), and (c_l, c_r), respectively, in addition to the checkered pattern.

We call maze-pairs fulfilling the above properties *feasible for c.* From this definition it follows that the optimal solution for the cell $c^* := (e_L, \perp, \emptyset, \top, \emptyset, \emptyset)$ is the weakly feasible k-thin maze pair (T_{c^*}, M_{c^*}) with maximum weight $w(T_{c^*})$.

We define a partial order $\prec$ for the cells and fill in the DP table w.r.t. this order (breaking ties arbitrarily). Intuitively speaking, we define $\prec$ to ensure that $c' \prec c''$ if the area (within the capacity curve) corresponding to c' is contained in the area corresponding to c''. The following definition achieves this: for two edges e' and e'', we let $|e' - e''|$ be the number of edges between e' and e'', boundary included. We define that $c' \prec c''$ if (in a lexicographic sense) $(|e_\uparrow(c') - e_\downarrow(c')|, |e(c') - e_\downarrow(c')|) <_{lex} (|e_\uparrow(c'') - e_\downarrow(c'')|, |e(c'') - e_\downarrow(c'')|)$.

The base case cells are obtained when $e = e_\downarrow$. In this case one must have $m_\uparrow = \top$, and hence $e_\uparrow = e$. Also $T(c) = \emptyset = M(c)$. For those cells we set $(T_c, M_c) := (\emptyset, \emptyset)$ (hence $w(c) = 0$).

Consider a cell c that is not a base case. For the sake of presentation, assume that $e_\downarrow$ is to the right of e, the other case being symmetric. Let e_r be the first edge to the right of e (possibly $e_r = e_\downarrow$). We will compute (T_c, M_c) as a function of some pairs $(T_{c'}, M_{c'})$ with $c' \prec c$, considering the following three branching cases (see Figure 5):

• **(single branching)** This case applies only when $m_\uparrow$ uses both e and e_r (possibly $m_\uparrow = \top$). Consider any feasible DP-cell $c_s = (e_r, m_\downarrow, C_\downarrow, m_\uparrow, C_\uparrow, B_s)$ with the following extra *compatibility property*: for $T' := C_\downarrow \cup C_\uparrow \cup B \cup B_s$, one has $crit(m_\downarrow, T') = C_\downarrow$, $crit(m_\uparrow, T') = C_\uparrow$, $bound_e(m_\downarrow, m_\uparrow, T') = B$, and $bound_{e_r}(m_\downarrow, m_\uparrow, T') = B_s$. Set $w_{sb}(c) \leftarrow$

$\max_{c_s}\{w(c_s) + w(B_s \setminus B)\}$.

• **(top–bottom branching)** This case applies only when $m_\uparrow$ uses both e and e_r (possibly $m_\uparrow = \top$). Consider any m-task $m_{mid} \neq \top$ that has e_r as its leftmost edge and such that $b(m_\downarrow) < b(m_{mid}) < b(m_\uparrow)$. Consider any pair of feasible entries $c_b = (e_r, m_\downarrow, C_\downarrow, m_{mid}, C_{mid}, B_b)$ and $c_t = (e_r, m_{mid}, C_{mid}, m_\uparrow, C_\uparrow, B_t)$ with the following extra *compatibility property*: for $T' := C_\downarrow \cup C_\uparrow \cup C_{mid} \cup B \cup B_b \cup B_t$, one has $crit(m_\uparrow, T') = C_\uparrow$, $crit(m_\downarrow, T') = C_\downarrow$, $crit(m_{mid}, T') = C_{mid}$, $bound_e(m_\downarrow, m_\uparrow, T') = B$, $bound_{e_r}(m_\downarrow, m_{mid}, T') = B_b$, and finally $bound_{e_r}(m_{mid}, m_\uparrow, T') = B_t$. Set $w_{tb}(c) \leftarrow \max_{(c_b,c_t)}\{w(c_b) + w(c_t) + w((B_b \cup B_t) \setminus B)\}$.

• **(left–right branching)** This branching applies only to the case that e is the rightmost edge of $m_\uparrow$, and $m_\uparrow \neq \top$. Consider any m-task m_{abv} that uses both e_r and e and with $b(m_{abv}) > b(m_\uparrow)$ (possibly $m_{abv} = \top$). Consider the pairs of feasible entries $c_l = (e, m_\uparrow, C_\uparrow, m_{abv}, C_{abv}, B_l)$ and $c_r = (e_r, m_\downarrow, C_\downarrow, m_{abv}, C_{abv}, B_r)$ with the following extra *compatibility property*: for $T' := C_\downarrow \cup C_\uparrow \cup C_{abv} \cup B \cup B_l \cup B_r$, one has $crit(m_\uparrow, T') = C_\uparrow$, $crit(m_\downarrow, T') = C_\downarrow$, $crit(m_{abv}, T') = C_{abv}$, $bound_e(m_\downarrow, m_\uparrow, T') = B$, $bound_{e_r}(m_\downarrow, m_{abv}, T') = B_r$, and $bound_e(m_\uparrow, m_{abv}, T') = B_l$. We set $w_{lr}(c) \leftarrow \max_{(c_l,c_r)}\{w(c_l) + w(c_r) + w((B_l \cup B_r) \setminus B)\}$.

Finally, we set $w(c) := \max\{w_{sb}(c), w_{tb}(c), w_{lr}(c)\}$. Depending on the case attaining the maximum, we define (T_c, M_c): if the maximum is achieved in the single-branching case for some c_s, then we set $T_c \leftarrow T_{c_s} \cup (B_s \setminus B)$ and $M_c \leftarrow M_{c_s}$. If the maximum is achieved in the top–bottom branching for some c_b and c_t, we set $T_c \leftarrow T_{c_b} \cup T_{c_t} \cup ((B_b \cup B_t) \setminus B)$ and $M_c \leftarrow M_{c_b} \cup M_{c_t} \cup \{m_{mid}\}$. Similarly, if the maximum is achieved in the left–right branching for some c_l and c_r, we set $T_c \leftarrow T_{c_l} \cup T_{c_r} \cup ((B_l \cup B_r) \setminus B)$ and $M_c \leftarrow M_{c_l} \cup M_{c_r} \cup \{m_{abv}\}$.

Observe that, in the single branching case, one has that $|e_\uparrow(c_s) - e_\downarrow(c_s)| = |e_\uparrow(c) - e_\downarrow(c)|$ and that $|e(c_s) - e_\downarrow(c_s)| < |e(c) - e_\downarrow(c)|$. In the other cases one has $|e_\uparrow(c') - e_\downarrow(c')| < |e_\uparrow(c) - e_\downarrow(c)|$, where $c' \in \{c_b, c_t, c_l, c_r\}$. Hence $c_s, c_b, c_t, c_l, c_r \prec c$ as required. Note also that c^* is the only feasible DP-cell associated to edge e_L and for any other DP-cell c it holds that $c \prec c^*$. The DP outputs (T_{c^*}, M_{c^*}) and we return T_{c^*} as the computed set of tasks.

Lemma 15. *The above dynamic program runs in time* $n^{O(k+1/\delta^2)}$.

Proof. Let us first bound the number of DP cells. Each DP cell is characterized by a tuple $(e, m_\uparrow, C_\uparrow, m_\downarrow, C_\downarrow, B)$. By the preprocessing step there are $O(n)$ choices for e, and by the definition of the m-tasks there are $O(n^2)$ choices for $m_\downarrow$ and $m_\uparrow$. Since $|C_\downarrow|, |C_\uparrow| \leq ncrit(\delta)$ and $|B| \leq k$ by definition, there are $n^{O(1/\delta^2)}$ choices for $C_\downarrow$ and $C_\uparrow$, and $O(n^k)$ choices for B. Next observe that in the DP, to compute the value of a DP-cell, one needs to consider any choice of at most two other DP-cells with certain restrictions, and to perform a polynomial number of operations for each such choice. The claim follows. □

Recall that by Lemma 3, to obtain a $1 + \varepsilon$ approximation for δ-large tasks, we need to choose $k = \Theta(\frac{1}{\varepsilon} + \frac{1}{\delta})$ which gives an overall running time of $n^{O(1/\varepsilon + 1/\delta^2)}$ for our DP. Note also that, from Theorem 1, to get a $1 + \varepsilon$ approximation for δ-small tasks, one has to choose $\delta = O(\varepsilon^2)$. Therefore, our $2 + \varepsilon$ approximation for UFP runs in time $n^{O(1/\varepsilon^4)}$. For a comparison, the running time of the $2 + \varepsilon$ approximation for UFP under the NBA in [17] is also $n^{O(1/\varepsilon^4)}$.

We next show the correctness of the dynamic program. Consider any cell c. First observe that $T_c \subseteq T(c)$ and $M_c \subseteq M(c)$. Also, by an easy induction, any two distinct m-tasks in M_c have different bottleneck capacity. In other terms, (T_c, M_c) is a well-defined maze pair. We next prove that $w(T_c) \geq w(T'_c)$ for any feasible pair (T'_c, M'_c) for c. To that aim, we prove that if a pair (T'_c, M'_c) is feasible for a cell c, then it can be decomposed into the feasible solution for a cell c_s and the tasks in $B(c_s) \setminus B$ or into feasible solutions for two cells c_t, c_b (or c_l, c_r) and the tasks in $(B(c_t) \cup B(c_b)) \setminus B$ (or $(B(c_l) \cup B(c_r)) \setminus B$), depending on the applying branching case.

Lemma 16. *If (T'_c, M'_c) is a feasible maze-pair for cell c, then $w(T'_c) \leq w(T_c)$.*

Proof. We show the claim by induction, following the partial order $\prec$ on cells. For the base cases, it is clear that $w(T'_c) = w(T_c) = 0$ and the claim follows.

Now consider a non-base-case cell c and suppose the claim is true for all cells c' with $c' \prec c$. W.l.o.g. assume again that $e_\downarrow$ lies on the right of e, and let e_r be the first edge to the right of e. We distinguish cases, depending on which m-tasks use e_r.

First suppose that there is no m-task $m_{mid} \in M'_c \cap M_{e_r}$ with $b(m_\downarrow) < b(m_{mid}) < b(m_\uparrow)$ using e_r and that $m_\uparrow$ uses e_r, where possibly $m_\uparrow = \top$ (single branching case). Then consider the DP cell $c_s = (e_r, m_\downarrow, C_\downarrow, m_\uparrow, C_\uparrow, B_s)$ with $B_s = bound_{e_r}(m_\downarrow, m_\uparrow, T'_c \cup B)$. Since (T'_c, M'_c) is feasible for c, c_s is indeed a cell in our DP table. In particular, observe that $|B_s| \leq k$ since (T'_c, M'_c) is k-thin. The consistency property follows by the weak feasibility of (T'_c, M'_c) and from the compatibility property of the single branching. By induction, we know that the DP computed the optimal solution (T_{c_s}, M_{c_s}) for c_s. In particular, $w(T_{c_s}) \geq w(T'_c) - w(B_s \setminus B)$ since $(T'_c \setminus (B_s \setminus B), M'_c)$ is feasible for c_s. By definition of the DP transition,

$$\begin{aligned} w(T_c) &\geq w_{sb}(c) \geq w(T_{c_s}) + w(B_s \setminus B) \\ &\geq (w(T'_c) - w(B_s \setminus B)) + w(B_s \setminus B) = w(T'_c). \end{aligned}$$

Next consider the case that there is an m-task $m_{mid} \in M'_c \cap M_{e_r}$ with $b(m_\downarrow) < b(m_{mid}) < b(m_\uparrow)$ using e_r. Note that by our preprocessing then $m_\uparrow$ uses e_r where possibly $m_\uparrow = \top$ (top–bottom branching). Also observe that there can be at most one such task m_{mid} by our preprocessing and using that any two m-tasks in a maze pair have different bottleneck capacities. Let us consider the (bottom) cell $c_b = (e_r, m_\downarrow, C_\downarrow, m_{mid}, C_{mid}, B_b)$ and the (top) cell $c_t = (e_r, m_{mid}, C_{mid}, m_\uparrow, C_\uparrow, B_t)$ where we define $B_b := bound_{e_r}(m_\downarrow, m_{mid}, T'_c \cup B)$, $B_t := bound_{e_r}(m_{mid}, m_\uparrow, T'_c \cup B)$, and $C_{mid} := crit(m_{mid}, T'_c \cup B)$. Also in this case, the feasibility of (T'_c, M'_c) for c implies that c_l and c_r are in fact DP cells. In particular, since (T'_c, M'_c) is weakly feasible, $|C_{mid}| \leq ncrit(\delta)$ by Lemma 7. The pair $(T'_c \cap T(c_b), M'_c \cap M(c_b)\})$ is feasible for c_b and the pair $(T'_c \cap T(c_t), M'_c \cap M(c_b))$ is feasible for c_t. In this case T'_c is partitioned by $T'_c \cap T(c_b)$, $T'_c \cap T(c_t)$, and $(B_b \cup B_t) \setminus B$. Hence,

$$\begin{aligned} w(T_c) &\geq w_{tb}(c) \geq w(c_b) + w(c_t) + w((B_b \cup B_t) \setminus B) \\ &\geq w(T'_c \cap T(c_b)) + w(T'_c \cap T(c_t)) + w((B_b \cup B_t) \setminus B) \\ &= w(T'_c). \end{aligned}$$

Finally, consider the case that there is no m-task $m_{mid} \in M'_c \cap M_{e_r}$ with $b(m_\downarrow) < b(m_{mid}) < b(m_\uparrow)$ and that $m_\uparrow$ does not use e_r (left–right branching case). Let $m_{abv} \in M'_c \cap M_{e_r}$ be the m-task minimizing $b(m_{abv})$ such that $b(m_{abv}) > b(m_\uparrow)$ (possibly $m_{abv} = \top$). By the preprocessing of the input tasks, if $m_{abv} \neq \top$, then m_{abv} must use e, as well (otherwise two m-tasks with different bottleneck capacities would share one endpoint).

Consider now the cells $c_l = (e, m_\uparrow, C_\uparrow, m_{abv}, C_{abv}, B_l)$ and $c_r = (e_r, m_\downarrow, C_\downarrow, m_{abv}, C_{abv}, B_r)$ where we define $B_l = bound_e(m_\uparrow, m_{abv}, T'_c \cup B)$, $B_r = bound_{e_r}(m_\downarrow, m_{abv}, T'_c \cup B)$, and $C_{abv} = crit(m_{abv}, T'_c \cup B)$. Again, since (T'_c, M'_c) is feasible for c, c_l and c_r are in fact DP cells.

Also, the pair $(T'_c \cap T(c_l), M'_c \cap M(c_l))$ is feasible for c_l and the pair $(T'_c \cap T(c_r), M'_c \cap M(c_r))$ is feasible for c_r. By induction, we know that the DP computed the optimal solutions (T_{c_l}, M_{c_l}) and (T_{c_r}, M_{c_r}) for c_l and c_r, respectively. Observe that T'_c is partitioned by $T'_c \cap T(c_l)$, $T'_c \cap T(c_r)$, and $(B_l \cup B_r) \setminus B$. Hence,

$$\begin{aligned} w(T_c) &\geq w_{lr}(c) \geq w(c_l) + w(c_r) + w((B_l \cup B_r) \setminus B) \\ &= w(T'_c \cap T(c_l)) + w(T'_c \cap T(c_r)) + w((B_l \cup B_r) \setminus B) \\ &= w(T'_c). \end{aligned}$$

This concludes the proof. □

We next prove that the DP computes a feasible solution (satisfying properties 1-3) for each DP cell c. The proofs of the next three lemmas use a similar inductive pattern. We show that whenever we extend the solution for a cell c_s or combine the solutions for two cells c_t, c_b or c_l, c_r to a solution for some cell c according to the DP transition, then the new solution is k-thin (has the inclusion property, is weakly feasible) assuming that the original cells c_s or c_t, c_b or c_l, c_r were k-thin (have the inclusion property, are weakly feasible).

Lemma 17 (k-thinness). *For each DP-cell $c = (e, m_\downarrow, C_\downarrow, m_\uparrow, C_\uparrow, B)$, we have that $(T_c \cup B, M_c \cup \{m_\downarrow, m_\uparrow\})$ is k-thin.*

Proof. It is sufficient to show that, given any edge e and any two m-tasks $m', m'' \in (M_c \cup \{m_\downarrow, m_\uparrow\}) \cap M_e$, with $b(m') < b(m'')$ and such that there is no $m''' \in (M_c \cup \{m_\downarrow, m_\uparrow\}) \cap M_e$ with $b(m') < b(m''') < b(m'')$, then the number of tasks i in $(T_c \cup B) \cap T_e$ with $b(m') < b(i) \leq b(m'')$ is at most k. In other terms, $|bound_e(m', m'', T_c \cup B)| \leq k$.

We prove the latter claim by induction, following the partial order $\prec$ on the cells. For the base cases, recall that for each DP cell c we required that $|B(c)| \leq k$. Hence, in that case $(T_c \cup B, M_c \cup \{m_\downarrow, m_\uparrow\}) = (B, \{m_\downarrow, m_\uparrow\})$ and the claim is trivially true.

Now consider a non-base-case DP cell c and suppose the claim is true for all cells c' with $c' \prec c$. Assume w.l.o.g. that $e_\downarrow$ lies on the right of e. We distinguish the three branching cases and show that in each case the pair (T_c, M_c) is k-thin.

First suppose that the single branching case applies, that is, there is a cell c_s such that $T_c = T_{c_s} \cup (B(c_s) \setminus B)$ and $M_c = M_{c_s}$. By induction $(T_{c_s} \cup B(c_s), M_{c_s} \cup \{m_\downarrow, m_\uparrow\})$ is k-thin. Hence, it suffices to ensure that $|bound_e(m_\downarrow, m_\uparrow, T_c \cup B)| \leq k$. However, the latter holds since $|bound_e(m_\downarrow, m_\uparrow, T_c \cup B)| = |bound_e(m_\downarrow, m_\uparrow, B)| = |B|$ by the compatibility property of the branching, and $|B| \leq k$ by the definition of DP cells.

The same basic argument also works for the remaining two branching cases: it is sufficient to bound $|bound_e(m_\downarrow, m_\uparrow, T_c \cup B)|$, and an upper bound of k follows from the compatibility property of the considered branching and by definition of DP cells. □

Lemma 18 (Inclusion property). *For each DP-cell $c = (e, m_\downarrow, C_\downarrow, m_\uparrow, C_\uparrow, B)$, if $i \in crit(m_\uparrow, T_c)$ then $i \in C_\uparrow$.*

Proof. We prove this claim by using the compatibility properties of the branching procedures. The claim is trivially true for base case cells c since $T_c \subseteq T(c) = \emptyset$.

Consider now a non-base-case cell c, and assume the claim holds for any cell $c' \prec c$. Assume w.l.o.g. that $e_\downarrow$ lies on the right of e. Suppose that $T_c = T_{c_s} \cup (B_s \setminus B)$ for some cell $c_s = (e_r, m_\downarrow, C_\downarrow, m_\uparrow, C_\uparrow, B_s)$ in the single branching case. If $i \in crit(m_\uparrow, B_s \setminus B)$ then $i \in C_\uparrow$ by the compatibility property of the single branching procedure. If $i \in crit(m_\uparrow, T_{c_s})$ then $i \in C_\uparrow$ by the induction hypothesis.

Assume now that $T_c = T_{c_t} \cup T_{c_b} \cup ((B(c_t) \cup B(c_b)) \setminus B)$ holds for two cells $c_b = (e_r, m_\downarrow, C_\downarrow, m_{mid}, C_{mid}, B_b)$ and $c_t = (e_r, m_{mid}, C_{mid}, m_\uparrow, C_\uparrow, B_t)$ in the top–bottom branching case. If $i \in crit(m_\uparrow, T_{c_t})$ then $i \in C_\uparrow$ by the induction hypothesis. If $i \in crit(m_\uparrow, T_{c_b})$ then $i \in crit(m_{mid}, T_{c_b})$ and hence $i \in C_{mid}$ by the induction hypothesis. Now the compatibility property of the top–bottom branching case implies that $i \in C_\uparrow$ (using that $i \in crit(m_\uparrow, T_{c_b})$). If $i \in crit(m_\uparrow, (B(c_t) \cup B(c_b)) \setminus B)$ then $i \in C_\uparrow$ by the compatibility property of the top–bottom branching case.

Finally, assume that $T_c = T_{c_l} \cup T_{c_r} \cup ((B_l \cup B_r) \setminus B)$ for two DP cells which are defined as $c_l = (e, m_\uparrow, C_\uparrow, m_{abv}, C_{abv}, B_l)$ and $c_r = (e_r, m_\downarrow, C_\downarrow, m_{abv}, C_{abv}, B_r)$ in the left–right branching case. If $i \in T_{c_l}$ then $b(i) > b(m_\uparrow)$, so i is not critical for $m_\uparrow$ and there is nothing to show. If $i \in crit(m_\uparrow, T_{c_r})$ then also $i \in B$ and the claim follows from the compatibility property of the left–right branching case. Finally, if $i \in ((B_l \cup B_r) \setminus B$ then the claim also follows from the compatibility property. □

Lemma 19 (Weak feasibility)**.** *For each DP-cell $c = (e, m_\downarrow, C_\downarrow, m_\uparrow, C_\uparrow, B)$, we have that $(T_c \cup B \cup C_\downarrow \cup C_\uparrow, M_c \cup \{m_\downarrow, m_\uparrow\})$ is weakly feasible.*

Proof. For any edge f, define $m_f := m_f(c)$ as the highest bottleneck capacity m-task in $M_c \cup \{m_\downarrow(c), m_\uparrow(c), \perp\} \setminus \{\top\}$ using edge f. Let also $T_c^{ext} := T_c \cup C_\downarrow(c) \cup C_\uparrow(c) \cup B(c)$. With this notation, we need to prove that for each edge f

$$d(abv_f(m_f(c), T_c^{ext})) + d(crit_f(m_f(c), T_c^{ext})) + d(m_f(c)) \leq u_f.$$

We prove the claim by induction, following the partial order $\prec$ on the cells. If c is a base case cell then $T_c \subseteq T(c) = \emptyset$, $M_c \subseteq M(c) = \emptyset$ and hence $T_c^{ext} = C_\downarrow \cup C_\uparrow \cup B$. By the consistency property, $(T_c^{ext}, \{m_\downarrow, m_\uparrow\})$ is weakly feasible.

For notation convenience, let us say that $e' < e''$ if edge e' is to the left of edge e'' and $e' \neq e''$. We define analogously $\leq$, $>$, and $\geq$.

Suppose now that c is not a base case cell. By induction hypothesis, we know that the claim holds for any cell $c' \prec c$. Assume w.l.o.g. that $e < e_\downarrow$. Let e_r be the first edge to the right of e (possibly $e_r = e_\downarrow$). We distinguish 3 cases, depending on the branching that defines the maximum value of $w(c)$.

a) (Single branching) Let c_s be the cell achieving the maximum. Recall that $T_c = T_{c_s} \cup (B_s \setminus B)$ and $M_c = M_{c_s}$. We have $m_f = m_f(c) = m_f(c_s)$ because $M_c = M_{c_s}$. Let us assume $e_\uparrow < e$, the case $e_\uparrow \geq e$ being analogous. Consider any edge f. We distinguish 3 subcases depending on the relative position of f:

a.1) ($f \leq e_\uparrow$ or $f \geq e_\downarrow$) Here $T(c) \cap T_f = M(c) \cap M_f = \emptyset$, hence $T_c^{ext} = C_\downarrow \cup C_\uparrow \cup B$ and $M_c \cup \{m_\downarrow, m_\uparrow\} = \{m_\downarrow, m_\uparrow\}$. The claim follows by the consistency property.

a.2) ($e < f < e_\downarrow$). In this range of edges we have $(B \setminus B_s) \cap T_f = \emptyset$ by the compatibility property of the single branching case. Hence

$$\begin{aligned} T_c^{ext} \cap T_f &= (T_c \cup C_\downarrow \cup C_\uparrow \cup B) \cap T_f \\ &= (T_{c_s} \cup B \cup B_s \cup C_\downarrow \cup C_\uparrow) \cap T_f \\ &= (T_{c_s} \cup C_\downarrow \cup C_\uparrow \cup B_s) \cap T_f = T_{c_s}^{ext} \cap T_f. \end{aligned}$$

As a consequence, $abv_f(m_f, T_c^{ext}) = abv_f(m_f, T_{c_s}^{ext})$ and $crit_f(m_f, T_c^{ext}) = crit_f(m_f, T_{c_s}^{ext})$. The claim follows by induction hypothesis on c_s.

a.3) ($e_\uparrow < f \le e$) In this case $b(m_f) \ge b(m_\uparrow)$. Since any task $i \in B$ has $b(i) \le b(m_\uparrow)$, we have

$$\begin{aligned} abv_f(m_f, T_c^{ext}) &= abv_f(m_f, T_{c_s} \cup B_s \cup B \cup C_\downarrow \cup C_\uparrow) \\ &= abv_f(m_f, T_{c_s} \cup B_s \cup C_\downarrow \cup C_\uparrow) \\ &= abv_f(m_f, T_{c_s}^{ext}). \end{aligned}$$

Also, any task $i \in B$ that is critical for m_f must be contained in $C_\uparrow$ by the compatibility property of the single branching, hence $crit_f(m_f, B) \subseteq C_\uparrow \cap T_f$. Therefore

$$\begin{aligned} crit_f(m_f, T_c^{ext}) &= crit_f(m_f, T_{c_s} \cup B_s \cup B \cup C_\downarrow \cup C_\uparrow) \\ &= crit_f(m_f, T_{c_s} \cup B_s \cup C_\downarrow \cup C_\uparrow) \\ &= crit_f(m_f, T_{c_s}^{ext}). \end{aligned}$$

The claim then follows by induction hypothesis on c_s.

b) (Top-bottom branching) Let c_b and c_t be the cells achieving the maximum. Recall that $M_c = M_{c_b} \cup M_{c_t} \cup \{m_{mid}\}$ and $T_c = T_{c_b} \cup T_{c_t} \cup ((B_b \cup B_t) \setminus B)$. Let $e_{mid} := e(m_{mid})$. Note that $e < e_{mid}$ and $e_\uparrow < e_{mid}$. Let us assume $e_\uparrow < e$, the case $e_\uparrow \ge e$ being analogous. Consider any edge f. We distinguish 4 subcases:

b.1) ($f \le e_\uparrow$ or $f \ge e_\downarrow$) Here $T(c) \cap T_f = M(c) \cap M_f = \emptyset$. The claim follows by the same argument as in case (a.1).

b.2) ($e_{\mathbf{mid}} \le f < e_\downarrow$) Note that $M(c_t) \cap M_f = T(c_t) \cap T_f = \emptyset$. We have $m_f = m_f(c) = m_f(c_b)$. Observe also that $((B \cup B_t) \setminus B_b) \cap T_f = \emptyset$ and $C_\uparrow \cap T_f \subseteq C_{mid} \cap T_f$ by the compatibility property of the top–bottom branching case. Altogether

$$\begin{aligned} T_c^{ext} \cap T_f &= (T_{c_b} \cup T_{c_t} \cup B_b \cup B_t \cup B \cup C_\uparrow \cup C_\downarrow) \cap T_f \\ &= (T_{c_b} \cup B_b \cup C_\uparrow \cup C_\downarrow) \cap T_f \\ &\subseteq (T_{c_b} \cup B_b \cup C_{mid} \cup C_\downarrow) \cap T_f = T_{c_b}^{ext} \cap T_f. \end{aligned}$$

Then $abv_f(m_f, T_c^{ext}) \subseteq abv_f(m_f, T_{c_b}^{ext})$ and $crit_f(m_f, T_c^{ext}) \subseteq crit_f(m_f, T_{c_b}^{ext})$. The claim follows by induction hypothesis on c_b.

b.3) ($e_{\mathbf{r}} \le f < e_{\mathbf{mid}}$) We have $m_f = m_f(c) = m_f(c_t)$ and observe that $b(m_f) \ge b(m_{mid})$. By the compatibility property of the top–bottom branching, if $i \in B \cap T_f$ and $b(i) > b(m_{mid})$, then $i \in B_t$. Note also that for any $i \in (C_{mid} \cup C_\downarrow \cup T_{c_b} \cup B_b) \cap T_f$ we have $b(i) \le b(m_{mid})$. Altogether

$$\begin{aligned} & abv_f(m_f, T_c^{ext}) \\ =& abv_f(m_f, T_{c_b} \cup T_{c_t} \cup B_b \cup B_t \cup B \cup C_\downarrow \cup C_\uparrow) \\ =& abv_f(m_f, T_{c_t} \cup B_t \cup C_\uparrow) \\ =& abv_f(m_f, T_{c_t} \cup B_t \cup C_{mid} \cup C_\uparrow) = abv_f(m_f, T_{c_t}^{ext}). \end{aligned}$$

By the compatibility property of the top–bottom branching, if $i \in (B_b \cup B \cup C_\downarrow) \cap T_f$ is critical for m_f (hence for m_{mid}), then $i \in C_{mid}$. By Lemma 18, if $i \in T_{c_b} \cap T_f$ is critical for m_f (hence for m_{mid}), then $i \in C_{mid}$. Altogether $crit_f(m_f, T_c^{ext}) = crit_f(m_f, T_{c_b} \cup T_{c_t} \cup B_b \cup B_t \cup B \cup C_\downarrow \cup C_\uparrow) \subseteq crit_f(m_f, T_{c_t} \cup B_t \cup C_\uparrow \cup C_{mid}) = crit_f(m_f, T_{c_t}^{ext})$. The claim follows by induction hypothesis on c_t.

b.4) ($\boldsymbol{e_\uparrow < f < e_r}$) We have $m_f = m_f(c) = m_f(c_t)$ and observe that $b(m_f) \geq b(m_\uparrow)$. Note that $T_{c_b} \cap T_f = \emptyset$. Also, for any $i \in (B \cup B_b \cup C_\downarrow \cup C_{mid}) \cap T_f$ we have that $b(i) \leq b(m_\uparrow)$. Then

$$\begin{aligned}&abv_f(m_f, T_c^{ext})\\ =&abv_f(m_f, T_{c_b} \cup T_{c_t} \cup B_b \cup B_t \cup B \cup C_\downarrow \cup C_\uparrow)\\ =&abv_f(m_f, T_{c_t} \cup B_t \cup C_\uparrow) = abv_f(m_f, T_{c_t}^{ext}).\end{aligned}$$

By the compatibility property of the top–bottom branching case, if $i \in (B_b \cup B \cup C_\downarrow \cup C_{mid}) \cap T_f$ is critical for m_f, then $i \in C_\uparrow$. Thus

$$\begin{aligned}&crit_f(m_f, T_c^{ext})\\ =&crit_f(m_f, T_{c_b} \cup T_{c_t} \cup B_b \cup B_t \cup B \cup C_\downarrow \cup C_\uparrow)\\ =&crit_f(m_f, T_{c_t} \cup B_t \cup C_\uparrow \cup C_{mid}) = crit_f(m_f, T_{c_t}^{ext}).\end{aligned}$$

The claim follows by induction hypothesis on c_t.

c) (Left-right branching) Let c_l and c_r be the cells achieving the maximum. Recall that $M_c = M_{c_l} \cup M_{c_r} \cup \{m_{abv}\}$ and $T_c = T_{c_l} \cup T_{c_r} \cup ((B_l \cup B_r) \setminus B)$. Let $e_{abv} := e(m_{abv})$ ($e_{abv} := e_r$ if $m_{abv} = \top$). Let us assume that $e_{abv} > e_r$, the case $e_{abv} \leq e_r$ being analogous. Consider any edge f. We distinguish 4 subcases:

c.1) ($\boldsymbol{f \leq e_\uparrow}$ or $\mathbf{f \geq e_\downarrow}$) In this case $T(c) \cap T_f = M(c) \cap M_f = \emptyset$. The claim follows by the same argument as in case (a.1).

c.2) ($\boldsymbol{e_{abv} \leq f < e_\downarrow}$) In this case $T(c_l) \cap T_f = M(c_l) \cap M_f = \emptyset$. As a consequence, $m_f(c) = m_f(c_r)$. Also, $((B \cup B_l) \setminus B_r) \cap T_f = \emptyset$ by the compatibility property of the left–right branching. Furthermore, $C_\uparrow \cap T_f \subseteq B_r \cup C_\downarrow$ and $C_{abv} \cap T_f \subseteq B_r \cup T_{c_r} \cup C_\downarrow$. Then

$$\begin{aligned}T_c^{ext} \cap T_f &= (T_{c_l} \cup T_{c_r} \cup B_l \cup B_r \cup B \cup C_\downarrow \cup C_\uparrow) \cap T_f\\ &= (T_{c_r} \cup B_r \cup C_\downarrow) \cap T_f\\ &= (T_{c_r} \cup B_r \cup C_\downarrow \cup C_{abv}) \cap T_f = T_{c_r}^{ext} \cap T_f.\end{aligned}$$

As a consequence $abv_f(m_f(c), T_c^{ext}) = abv_f(m_f(c_r), T_{c_r}^{ext})$ and also $crit_f(m_f(c), T_c^{ext}) = crit_f(m_f(c_r), T_{c_r}^{ext})$. The claim follows by induction hypothesis on c_r.

c.3) ($\boldsymbol{e_r \leq f < e_{abv}}$) We have $m_f = m_f(c) = m_f(c_l)$ and $b(m_f) \geq b(m_{abv})$. Note that any task $i \in (T_{c_r} \cup B \cup B_r \cup C_\downarrow \cup C_{abv}) \cap T_f$ has $b(i) \leq b(m_{abv})$. Hence

$$\begin{aligned}&abv_f(m_f, T_c^{ext})\\ =&abv_f(m_f, T_{c_l} \cup T_{c_r} \cup B_l \cup B_r \cup B \cup C_\downarrow \cup C_\uparrow)\\ =&abv_f(m_f, T_{c_l} \cup B_l \cup C_\uparrow \cup C_{abv}) = abv_f(m_f, T_{c_l}^{ext}).\end{aligned}$$

Furthermore, if a task $i \in (T_{c_r} \cup B \cup B_r \cup C_\downarrow) \cap T_f$ is critical for m_f, then $i \in C_{abv}$ by the compatibility property of the left–right branching. Consequently

$$\begin{aligned}&crit_f(m_f, T_c^{ext})\\ =&crit_f(m_f, T_{c_l} \cup T_{c_r} \cup B_l \cup B_r \cup B \cup C_\downarrow \cup C_\uparrow)\\ \subseteq&crit_f(m_f, T_{c_l} \cup C_{abv} \cup C_\uparrow \cup B_l) = crit_f(m_f, T_{c_l}^{ext}).\end{aligned}$$

The claim follows by induction hypothesis on c_l.

c.4) ($\boldsymbol{e_\uparrow < f < e_r}$) In this case $m_f = m_f(c) = m_f(c_l)$ and $b(m_f) \geq b(m_\uparrow)$. By the compatibility property of the left–right branching, $((B_r \cup B) \setminus B_l) \cap T_f = B \cap T_f$. Observe that any task $i \in (B \cup C_\downarrow) \cap T_f$ has $b(i) \leq b(m_\uparrow)$. Also, $T_{c_r} \cap T_f \subseteq T(c_r) \cup T_f = \emptyset$. Then

$$\begin{aligned}
&abv_f(m_f, T_c^{ext})\\
=&abv_f(m_f, T_{c_l} \cup T_{c_r} \cup B_l \cup B_r \cup B \cup C_\downarrow \cup C_\uparrow)\\
=&abv_f(m_f, T_{c_l} \cup B_l \cup C_\uparrow)\\
\subseteq&abv_f(m_f, T_{c_l} \cup B_l \cup C_\uparrow \cup C_{abv}) = abv_f(m_f, T_{c_l}^{ext}).
\end{aligned}$$

By the compatibility property of the left–right branching, any $i \in (B \cup C_\downarrow) \cap T_f$ that is critical for m_f must belong to $C_\uparrow$. Thus

$$\begin{aligned}
&crit_f(m_f, T_c^{ext})\\
=&crit_f(m_f, T_{c_l} \cup T_{c_r} \cup B_l \cup B_r \cup B \cup C_\downarrow \cup C_\uparrow)\\
=&crit_f(m_f, T_{c_l} \cup B_l \cup C_\uparrow)\\
\subseteq&crit_f(m_f, T_{c_l} \cup B_l \cup C_\uparrow \cup C_{abv}) = crit_f(m_f, T_{c_l}^{ext}).
\end{aligned}$$

The claim follows by induction hypothesis on c_l. □

Now the proof of Lemma 6 follows from Lemmas 15-19, and the fact that the cell $c^* := (e^*, \perp, \emptyset, \top, \emptyset, \emptyset)$ corresponds to the optimal weakly-feasible k-thin maze-pair.

REFERENCES

[1] A. Adamaszek, P. Chalermsook, A. Ene, and A. Wiese. Submodular unsplittable flow on trees. In *IPCO*, volume 9682 of *Lecture Notes in Computer Science*, pages 337–349, 2016.

[2] A. Anagnostopoulos, F. Grandoni, S. Leonardi, and A. Wiese. Constant integrality gap LP formulations of unsplittable flow on a path. In *IPCO*, pages 25–36, 2013.

[3] A. Anagnostopoulos, F. Grandoni, S. Leonardi, and A. Wiese. A mazing 2+ε approximation for unsplittable flow on a path. In *SODA*, pages 26–41, 2014.

[4] E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18:1–8, 1987.

[5] Y. Azar and O. Regev. Combinatorial algorithms for the unsplittable flow problem. *Algorithmica*, 44:49–66, 2006.

[6] N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *STOC*, pages 721–729, 2006.

[7] N. Bansal, Z. Friggstad, R. Khandekar, and R. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *SODA*, pages 702–709, 2009.

[8] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *STOC*, pages 735–744, 2000.

[9] A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. In *STOC*, pages 735–744, 2000.

[10] R. Bar-Yehuda, M. Beder, Y. Cohen, and D. Rawitz. Resource allocation in bounded degree trees. In *ESA*, pages 64–75, 2006.

[11] J. Batra, N. Garg, A. Kumar, T. Mömke, and A. Wiese. New approximation schemes for unsplittable flow on a path. In *SODA*, pages 47–58, 2015.

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

[13] G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *IPCO*, pages 401–414, 2002.

[14] A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47(1):53–78, 2007.

[15] C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. Unpublished. Available at http://cs-people.bu.edu/aene/papers/ufp-full.pdf.

[16] C. Chekuri, A. Ene, and N. Korula. Unsplittable flow in paths and trees and column-restricted packing integer programs. In *APPROX-RANDOM*, pages 42–55, 2009.

[17] C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3, 2007. *An extended abstract appeared in the proceedings of the 30th International Colloquium on Automata, Languages and Programming (ICALP 2003).*

[18] B. Chen, R. Hassin, and M. Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34:501–507, 2002.

[19] M. Chrobak, G. Woeginger, K. Makino, and H. Xu. Caching is hard, even in the fault model. In *ESA*, pages 195–206, 2010.

[20] A. Darmann, U. Pferschy, and J. Schauer. Resource allocation with time intervals. *Theoretical Computer Science*, 411:4217–4234, 2010.

[21] 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.

[22] F. Grandoni, T. Mömke, A. Wiese, and H. Zhou. To augment or not to augment: Solving unsplittable flow on a path by creating slack. In *SODA*, pages 2411–2422, 2017.

[23] F. Grandoni, T. Mömke, A. Wiese, and H. Zhou. Unsplittable flow on a path: Breaking the 2-approximation barrier. 2017. unpublished.

[24] J. M. Kleinberg. *Approximation Algorithms for Disjoint Paths Problems*. PhD thesis, MIT, 1996.

[25] S. Leonardi, A. Marchetti-Spaccamela, and A. Vitaletti. Approximation algorithms for bandwidth and storage allocation problems under real time constraints. In *FSTTCS*, pages 409–420, 2000.

[26] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *SODA*, pages 879–888, 2000.

[27] C. A. Phillips, R. N. Uma, and J. Wein. Off-line admission control for general scheduling problems. In *SODA*, pages 879–888, 2000.

[28] A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, Berlin, 2003.