# Submodular Unsplittable Flow on Trees

Anna Adamaszek

University of Copenhagen, Denmark

13.09.2017

Joint work with Parinya Chalermsook, Alina Ene, and Andreas Wiese.

# Preliminaries

## Unsplittable Flow on Trees (UFP-tree)

- Input:
  - Undirected tree $T = (V, E)$ with edge capacities $u : E \to \mathbb{Z}_+$.
  - Set of tasks $\mathcal{T}$; each task $i \in \mathcal{T}$ has a start vertex $s_i \in V$, and end vertex $t_i \in V$, a demand $d_i \in \mathbb{Z}_+$ and a profit $w_i \in \mathbb{Z}_+$.

# Preliminaries

## Unsplittable Flow on Trees (UFP-tree)

- Input:
  - Undirected tree $T = (V, E)$ with edge capacities $u : E \to \mathbb{Z}_+$.
  - Set of tasks $\mathcal{T}$; each task $i \in \mathcal{T}$ has a start vertex $s_i \in V$, and end vertex $t_i \in V$, a demand $d_i \in \mathbb{Z}_+$ and a profit $w_i \in \mathbb{Z}_+$.
- Feasible solution: subset of the tasks $\mathcal{T}' \subseteq \mathcal{T}$ satisfying the capacity constraints for each edge.
- Goal: find a feasible solution maximizing the profit.

# Preliminaries

## Unsplittable Flow on Trees (UFP-tree)

- Input:
  - Undirected tree $T = (V, E)$ with edge capacities $u : E \to \mathbb{Z}_+$.
  - Set of tasks $\mathcal{T}$; each task $i \in \mathcal{T}$ has a start vertex $s_i \in V$, and end vertex $t_i \in V$, a demand $d_i \in \mathbb{Z}_+$ and a profit $w_i \in \mathbb{Z}_+$.
- Feasible solution: subset of the tasks $\mathcal{T}' \subseteq \mathcal{T}$ satisfying the capacity constraints for each edge.
- Goal: find a feasible solution maximizing the profit.

## Submodular Unsplittable Flow on Trees

Generalization of UFP-tree; instead of a linear weight function $w$ we have a submodular objective function $f : 2^{\mathcal{T}} \to \mathbb{R}_+$.

# Preliminaries

A function $f : 2^{\mathcal{T}} \rightarrow \mathbb{R}_+$ is *submodular* if $f(A) + f(B) \geq f(A \cap B) + f(A \cup B)$ for any two subsets $A, B \subseteq \mathcal{T}$.

We assume that $f$ is given as a value oracle, i.e., we are given access to an oracle that takes as input any set $S$ and outputs $f(S)$.

# Preliminaries

As the Unsplittable Flow on Trees problem is NP-hard, research is focused on finding good approximation algorithms for it.

An $\alpha$-*approximation algorithm* is a polynomial-time algorithm that for any input instance finds a solution with a value within an $\alpha$ factor of the value of an optimal solution.

# Results

Previous results:

- constant factor approximation for a path with linear objective,
- $O(\log^2 n)$-approximation for a tree with linear objective,
- $O(\log n)$-approximation for a path with submodular objective.

# Results

Previous results:

- constant factor approximation for a path with linear objective,
- $O(\log^2 n)$-approximation for a tree with linear objective,
- $O(\log n)$-approximation for a path with submodular objective.

**Theorem (A.,Chalermsook, Ene, Wiese; 2016)**

*There is a $O(k \cdot \log n)$ approximation for Submodular UFP on trees, where $k$ is the pathwidth of the tree and $n$ is the number of nodes in the tree.*

# Results

Previous results:

- constant factor approximation for a path with linear objective,
- $O(\log^2 n)$-approximation for a tree with linear objective,
- $O(\log n)$-approximation for a path with submodular objective.

**Theorem (A.,Chalermsook, Ene, Wiese; 2016)**

*There is a $O(k \cdot \log n)$ approximation for Submodular UFP on trees, where $k$ is the pathwidth of the tree and $n$ is the number of nodes in the tree.*

As each tree has pathwidth $O(\log n)$, this gives an $O(\log^2 n)$-approximation for arbitrary trees, matching the best known result for linear objective functions.

# High-level idea

1. UFP for linear objective, and polynomially bounded capacities
   - reduction to intersecting instances,
   - partitioning the tree into paths,
   - geometric viewpoint: drawing of tasks as rectangles below the capacity profile,
   - LP relaxation enforcing the geometric viewpoint,
   - rounding (randomized rounding with alteration strategy)

# High-level idea

1. UFP for linear objective, and polynomially bounded capacities
   - reduction to intersecting instances,
   - partitioning the tree into paths,
   - geometric viewpoint: drawing of tasks as rectangles below the capacity profile,
   - LP relaxation enforcing the geometric viewpoint,
   - rounding (randomized rounding with alteration strategy)
2. removing the "polynomially bounded" restriction
   - in the geometric viewpoint we allow only a polynomial number of placements for each task

# High-level idea

1. UFP for linear objective, and polynomially bounded capacities
   - reduction to intersecting instances,
   - partitioning the tree into paths,
   - geometric viewpoint: drawing of tasks as rectangles below the capacity profile,
   - LP relaxation enforcing the geometric viewpoint,
   - rounding (randomized rounding with alteration strategy)
2. removing the "polynomially bounded" restriction
   - in the geometric viewpoint we allow only a polynomial number of placements for each task
3. allowing submodular objective function
   - using contention resolution (CR) scheme

# Reduction to *intersecting instances*

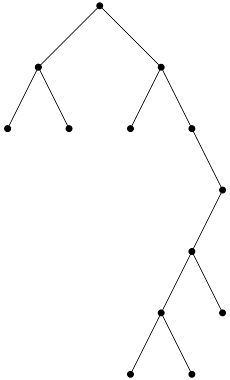
Intersecting instance: the path of each task contains the root of the tree.

**Lemma (Chekuri, Ene, Korula; 2009)**

*If there is an $\alpha$-approximation algorithm for UFP-tree on intersecting instances, there is a $O(\alpha \cdot \log n)$-approximation algorithm for arbitrary trees.*
*This holds also for the generalization of the problem in which the objective function is sub-additive, i.e., $f(A \cup B) \leq f(A) + f(B)$ for any two disjoint sets $A$ and $B$.*

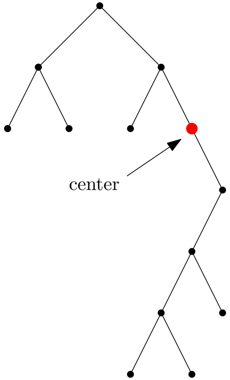
# Reduction to *intersecting instances*

idea:
partition the tasks into $O(\log n)$ sets; each set is a collection of independent intersecting instances



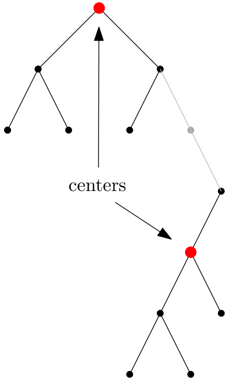
# Reduction to *intersecting instances*

idea:
partition the tasks into $O(\log n)$ sets; each set is a collection of independent intersecting instances



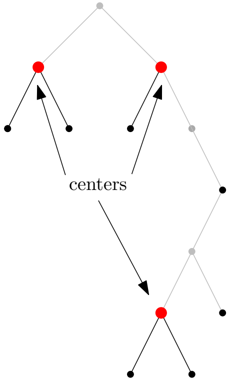
# Reduction to *intersecting instances*

idea:
partition the tasks into $O(\log n)$ sets; each set is a collection of independent intersecting instances



# Reduction to *intersecting instances*

idea:
partition the tasks into $O(\log n)$ sets; each set is a collection of independent intersecting instances



# Partitioning the tree $T$ into paths

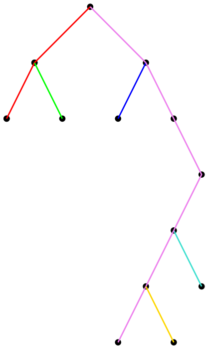
*upward path*: a path $p$ in a rooted tree $T$ s.t. one endpoint of $p$ is an ancestor in $T$ of the other endpoint

# Partitioning the tree $T$ into paths

*upward path*: a path $p$ in a rooted tree $T$ s.t. one endpoint of $p$ is an ancestor in $T$ of the other endpoint

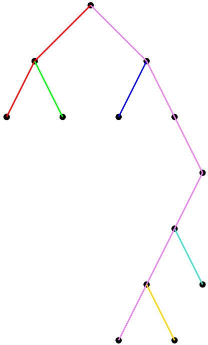
A collection of paths $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ in a rooted tree $T$ is a $K$-*nice splitting*, if

- the paths in $\mathcal{P}$ are edge-disjoint, upward paths, partitioning the edges of $T$, and
- each path in $T$ from a leaf to the root uses an edge of at most $K$ paths in $\mathcal{P}$.



# Partitioning the tree $T$ into paths

*upward path*: a path $p$ in a rooted tree $T$ s.t. one endpoint of $p$ is an ancestor in $T$ of the other endpoint



A collection of paths $\mathcal{P} = \{P_1, \ldots, P_\ell\}$ in a rooted tree $T$ is a $K$*-nice splitting*, if

- the paths in $\mathcal{P}$ are edge-disjoint, upward paths, partitioning the edges of $T$, and
- each path in $T$ from a leaf to the root uses an edge of at most $K$ paths in $\mathcal{P}$.

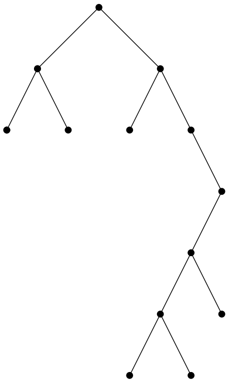
**Lemma**

*Let $T$ be a tree of pathwidth $k$. There is a polynomial time algorithm that constructs an $O(k)$-nice splitting for $T$.*

# Partitioning the tree $T$ into paths

Algorithm idea:

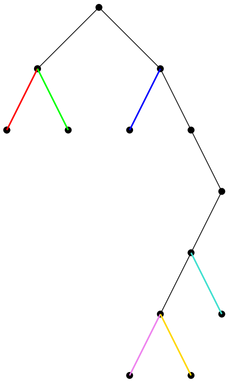
- process tree edges bottom-up, assigning colors (each color yields one path),
- an edge incident to a leaf gets a unique color,
- any other edge gets the same color as one of its children,
- the color is chosen to minimize the maximum number of colors on a leaf-to-root path



# Partitioning the tree $T$ into paths

Algorithm idea:

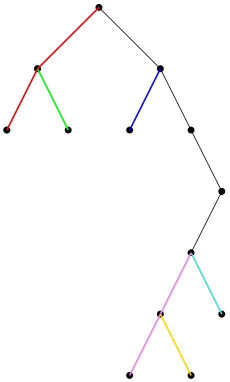
- process tree edges bottom-up, assigning colors (each color yields one path),
- an edge incident to a leaf gets a unique color,
- any other edge gets the same color as one of its children,
- the color is chosen to minimize the maximum number of colors on a leaf-to-root path



# Partitioning the tree $T$ into paths

Algorithm idea:

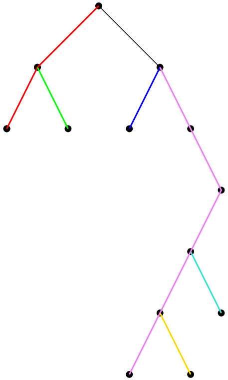
- process tree edges bottom-up, assigning colors (each color yields one path),
- an edge incident to a leaf gets a unique color,
- any other edge gets the same color as one of its children,
- the color is chosen to minimize the maximum number of colors on a leaf-to-root path



# Partitioning the tree $T$ into paths

Algorithm idea:

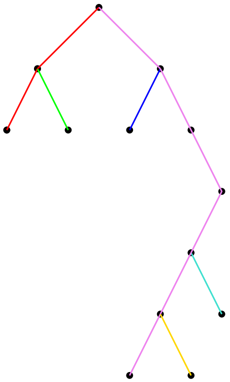
- process tree edges bottom-up, assigning colors (each color yields one path),
- an edge incident to a leaf gets a unique color,
- any other edge gets the same color as one of its children,
- the color is chosen to minimize the maximum number of colors on a leaf-to-root path



# Partitioning the tree $T$ into paths

Algorithm idea:

- process tree edges bottom-up, assigning colors (each color yields one path),
- an edge incident to a leaf gets a unique color,
- any other edge gets the same color as one of its children,
- the color is chosen to minimize the maximum number of colors on a leaf-to-root path



# Geometric viewpoint

intersecting instance of UFP on a rooted tree $T$,
$\mathcal{P} = \{P_1, \ldots, P_\ell\}$ – an $O(k)$-nice splitting for $T$

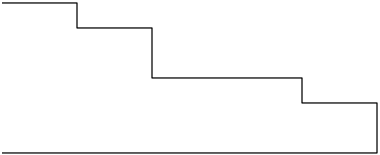
We create an instance of UFP on a *path* $P \in \mathcal{P}$: for each task $i \in \mathcal{T}$ corresponding to some path $p_i$ in $T$, and such that $p_i \cap P \neq \emptyset$, create a task corresponding to $p_i \cap P$.

# Geometric viewpoint

intersecting instance of UFP on a rooted tree $T$,
$\mathcal{P} = \{P_1, \ldots, P_\ell\}$ – an $O(k)$-nice splitting for $T$

We create an instance of UFP on a *path* $P \in \mathcal{P}$: for each task $i \in \mathcal{T}$ corresponding to some path $p_i$ in $T$, and such that $p_i \cap P \neq \emptyset$, create a task corresponding to $p_i \cap P$.

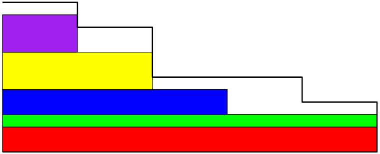
Observation: For a task $i$ and upward path $P$, if $i$ uses an edge of $P$ then it uses the top edge of $P$. We can assume w.l.o.g. that edge capacities on $P$ are non-increasing, i.e., $P$ is a *one-sided staircase*.



# Geometric viewpoint

## Lemma

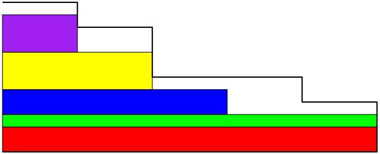
*Consider an instance of UFP on a path $P$, in which all tasks use the first edge of $P$. Any feasible subset of the tasks admits a representing drawing, i.e., it can be represented as a collection of non-overlapping rectangles drawn underneath the capacity profile, such that each task $i$ has a corresponding rectangle of height $d_i$ whose projection on $P$ is the path of $i$.*



# Geometric viewpoint

**Lemma**

*Consider an instance of UFP on a path $P$, in which all tasks use the first edge of $P$. Any feasible subset of the tasks admits a representing drawing, i.e., it can be represented as a collection of non-overlapping rectangles drawn underneath the capacity profile, such that each task $i$ has a corresponding rectangle of height $d_i$ whose projection on $P$ is the path of $i$.*



Proof idea: order the tasks in non-increasing order with respect to length, draw them one by one, as low as possible.

# Integer program

- We have a $O(k)$-nice splitting of the tree $T$ into paths $\mathcal{P}$.
- In the IP we will enforce that for every path $P \in \mathcal{P}$ there is a representing drawing (capacity constraints will be automatically satisfied).

# Integer program

- We have a $O(k)$-nice splitting of the tree $T$ into paths $\mathcal{P}$.
- In the IP we will enforce that for every path $P \in \mathcal{P}$ there is a representing drawing (capacity constraints will be automatically satisfied).

variables:

- $\forall i \in \mathcal{T}$, $x_i \in \{0, 1\}$ $\quad (x_i = 1$ if task $i$ is in the solution)
- $\forall P \in \mathcal{P}, i \in \mathcal{T}_{\mathcal{P}}$, $\forall h$ – allowed height for $i$, $y(i, h, P) \in \{0, 1\}$ $(y(i, h, P) = 1$ if task $i$ can be drawn at height $h$ for $P)$

# Integer program

- We have a $O(k)$-nice splitting of the tree $T$ into paths $\mathcal{P}$.
- In the IP we will enforce that for every path $P \in \mathcal{P}$ there is a representing drawing (capacity constraints will be automatically satisfied).

variables:

- $\forall i \in \mathcal{T}$, $x_i \in \{0,1\}$ $\quad$ ($x_i = 1$ if task $i$ is in the solution)
- $\forall P \in \mathcal{P}, i \in \mathcal{T}_\mathcal{P}$, $\forall h$ – allowed height for $i$, $y(i,h,P) \in \{0,1\}$ ($y(i,h,P) = 1$ if task $i$ can be drawn at height $h$ for $P$)

$$
\begin{aligned}
IP: \quad & \max \sum_{i \in \mathcal{T}} w_i \cdot x_i \\
\text{s.t.} \quad & \sum_{h-\text{allowed for } i \text{ at } P} y(i,h,P) = x_i \quad \forall P \in \mathcal{P}\ \forall i \in \mathcal{T}_\mathcal{P} \\
& \sum_{i \in \mathcal{T}_\mathcal{P}} \sum_{h-d_i < h' \le h} y(i,h',P) \le 1 \quad \forall P \in \mathcal{P}\ \forall h \le \max_{e \in P} u_e
\end{aligned}
$$

# LP relaxation

$$LP: \quad \max \sum_{i \in \mathcal{T}} w_i \cdot x_i$$

$$\text{s.t.} \sum_{h-\text{allowed for } i \text{ at } P} y(i,h,P) = x_i \quad \forall P \in \mathcal{P}\ \forall i \in \mathcal{T}_{\mathcal{P}}$$

$$\sum_{i \in \mathcal{T}_{\mathcal{P}}} \sum_{h-d_i < h' \leq h} y(i,h',P) \leq 1 \quad \forall P \in \mathcal{P}\ \forall h \leq \max_{e \in P} u_e$$

$$0 \leq x_i \leq 1, 0 \leq y(i,h,P) \leq 1$$

# LP relaxation

$$LP: \quad \max \sum_{i \in \mathcal{T}} w_i \cdot x_i$$

$$\text{s.t.} \sum_{h-\text{allowed for } i \text{ at } P} y(i,h,P) = x_i \quad \forall P \in \mathcal{P}\ \forall i \in \mathcal{T}_{\mathcal{P}}$$

$$\sum_{i \in \mathcal{T}_{\mathcal{P}}} \sum_{h-d_i < h' \leq h} y(i,h',P) \leq 1 \quad \forall P \in \mathcal{P}\ \forall h \leq \max_{e \in P} u_e$$

$$0 \leq x_i \leq 1, 0 \leq y(i,h,P) \leq 1$$

Randomized rounding with alteration

- selection phase: pick a subset of the tasks and determine a drawing for them (overlapping rectangles allowed)
- alteration phase: pick a subset of the selected tasks whose corresponding rectangles do not overlap

# Selection phase

We will create a (not necessarily feasible) set $S$ of tasks.

- Fix a large constant $c_1$.

# Selection phase

We will create a (not necessarily feasible) set $S$ of tasks.

- Fix a large constant $c_1$.
- For each task $i$, add $i$ to $S$ independently at random with probability $x_i/(c_1 \cdot k)$.

# Selection phase

We will create a (not necessarily feasible) set $S$ of tasks.

- Fix a large constant $c_1$.
- For each task $i$, add $i$ to $S$ independently at random with probability $x_i/(c_1 \cdot k)$.
- For each task $i \in S$ and path $P \in \mathcal{P}$ such that $i \in \mathcal{T}_P$, choose a height $h$ independently at random according to the probability distribution $y(i, h, P)/x_i$.

# Alteration phase

We will select a subset $S' \subseteq S$ of the tasks such that the corresponding rectangles are non-overlapping.

- Consider the paths of $\mathcal{P}$ in an arbitrary order.

# Alteration phase

We will select a subset $S' \subseteq S$ of the tasks such that the corresponding rectangles are non-overlapping.

- Consider the paths of $\mathcal{P}$ in an arbitrary order.
- For each $P \in \mathcal{P}$, let $S(P) = \{i \in S : i \in \mathcal{T}_P\}$.

# Alteration phase

We will select a subset $S' \subseteq S$ of the tasks such that the corresponding rectangles are non-overlapping.

- Consider the paths of $\mathcal{P}$ in an arbitrary order.
- For each $P \in \mathcal{P}$, let $S(P) = \{i \in S : i \in \mathcal{T}_P\}$.
- Goal: choose a set of *accepted tasks* $S'(P) \subseteq S(P)$, such that their rectangles are non-overlapping.

# Alteration phase

We will select a subset $S' \subseteq S$ of the tasks such that the corresponding rectangles are non-overlapping.

- Consider the paths of $\mathcal{P}$ in an arbitrary order.
- For each $P \in \mathcal{P}$, let $S(P) = \{i \in S : i \in \mathcal{T}_P\}$.
- Goal: choose a set of *accepted tasks* $S'(P) \subseteq S(P)$, such that their rectangles are non-overlapping.
- Order the tasks in $S(P)$ in *non-increasing* order according to their demands, breaking ties arbitrarily. Consider the tasks in this order.

# Alteration phase

We will select a subset $S' \subseteq S$ of the tasks such that the corresponding rectangles are non-overlapping.

- Consider the paths of $\mathcal{P}$ in an arbitrary order.
- For each $P \in \mathcal{P}$, let $S(P) = \{i \in S : i \in \mathcal{T}_P\}$.
- Goal: choose a set of *accepted tasks* $S'(P) \subseteq S(P)$, such that their rectangles are non-overlapping.
- Order the tasks in $S(P)$ in *non-increasing* order according to their demands, breaking ties arbitrarily. Consider the tasks in this order.
- Let $i$ be the current task. We add $i$ to $S'(P)$ if its rectangle does not overlap with any of the rectangles for the tasks we have accepted so far.

# Alteration phase

We will select a subset $S' \subseteq S$ of the tasks such that the corresponding rectangles are non-overlapping.

- Consider the paths of $\mathcal{P}$ in an arbitrary order.
- For each $P \in \mathcal{P}$, let $S(P) = \{i \in S : i \in \mathcal{T}_P\}$.
- Goal: choose a set of *accepted tasks* $S'(P) \subseteq S(P)$, such that their rectangles are non-overlapping.
- Order the tasks in $S(P)$ in *non-increasing* order according to their demands, breaking ties arbitrarily. Consider the tasks in this order.
- Let $i$ be the current task. We add $i$ to $S'(P)$ if its rectangle does not overlap with any of the rectangles for the tasks we have accepted so far.
- Let $S' = \{i \in S : \forall_{P \in \mathcal{P} : i \in \mathcal{T}_P}\ i \in S'(P)\}$, i.e., a task is accepted if it was accepted for all paths.

# Algorithm analysis

**Lemma**

- *For any path $P$ and task $i \in \mathcal{T}_P$, it holds that* $\Pr[i \notin S'(P) \mid i \in S(P)] \leq 2/(c_1 \cdot k)$.
- *Each selected task is rejected in the alteration phase with probability at most* $1/2$.

# Algorithm analysis

**Lemma**

- *For any path $P$ and task $i \in \mathcal{T}_P$, it holds that* $\Pr[i \notin S'(P) \mid i \in S(P)] \leq 2/(c_1 \cdot k)$.
- *Each selected task is rejected in the alteration phase with probability at most* $1/2$.

Proof idea: If a rectangle $R$ for the current task $i$ overlaps with some other rectangle, then it overlaps in the top left or the bottom left corner of $R$. This allows us to check the constraints only at two points.
The second property follows from the union bound.

## Lemma

- *For any path $P$ and task $i \in \mathcal{T}_P$, it holds that* $\Pr[i \notin S'(P) \mid i \in S(P)] \leq 2/(c_1 \cdot k)$.
- *Each selected task is rejected in the alteration phase with probability at most* $1/2$.

Proof idea: If a rectangle $R$ for the current task $i$ overlaps with some other rectangle, then it overlaps in the top left or the bottom left corner of $R$. This allows us to check the constraints only at two points.
The second property follows from the union bound.

## Theorem

*There is a $O(k \log n)$-approximation algorithm for UFP-tree on trees with pathwidth $k$ for linear objective functions and polynomially bounded edge capacities.*

# Removing restriction on edge capacities

We construct a polynomial-size set $\mathcal{H}$ of *allowed heights*, and we restrict the LP to place the tasks only at the heights from $\mathcal{H}$.

## Lemma

*For each feasible integral solution $\mathcal{T}' \subseteq \mathcal{T}$, there is a feasible fractional solution $(x, y)$ for the Restricted LP s.t. $\forall_{i \in \mathcal{T}'} \;\; x_i = \frac{1}{64}$.*

# Removing restriction on edge capacities

We construct a polynomial-size set $\mathcal{H}$ of *allowed heights*, and we restrict the LP to place the tasks only at the heights from $\mathcal{H}$.

**Lemma**

*For each feasible integral solution $\mathcal{T}' \subseteq \mathcal{T}$, there is a feasible fractional solution $(x, y)$ for the Restricted LP s.t. $\forall_{i \in \mathcal{T}'} \;\; x_i = \frac{1}{64}$.*

Idea:

- Each task can be placed as high as possible below the capacity profile (i.e., height $h = \min_{e \in P \cap p_i} u_e - d_i$ is in $\mathcal{H}$).
- We partition $\mathcal{T}$ into polynomially many *size classes*, and for each size class we add a polynomial number of heights for placing the tasks.

# Submodular objective: CR scheme

- $N$ – a finite ground set (tasks $\mathcal{T}$)

# Submodular objective: CR scheme

- $N$ – a finite ground set (tasks $\mathcal{T}$)
- $\mathcal{I} \subseteq 2^N$ – a family of subsets of $N$ (integral solutions)

# Submodular objective: CR scheme

- $N$ – a finite ground set (tasks $\mathcal{T}$)
- $\mathcal{I} \subseteq 2^N$ – a family of subsets of $N$ (integral solutions)
- $\mathbf{P}_{\mathcal{I}}$ – a convex relaxation for the constraints imposed by $\mathcal{I}$ (fractional solutions)

# Submodular objective: CR scheme

- $N$ – a finite ground set (tasks $\mathcal{T}$)
- $\mathcal{I} \subseteq 2^N$ – a family of subsets of $N$ (integral solutions)
- $\mathbf{P}_{\mathcal{I}}$ – a convex relaxation for the constraints imposed by $\mathcal{I}$ (fractional solutions)
  - $\mathbf{P}_{\mathcal{I}}$ is *down-monotone*: for $z, z' \in [0,1]^N$, if $z \leq z'$ and $z' \in \mathbf{P}_{\mathcal{I}}$, then $z \in \mathbf{P}_{\mathcal{I}}$

# Submodular objective: CR scheme

- $N$ – a finite ground set (tasks $\mathcal{T}$)
- $\mathcal{I} \subseteq 2^N$ – a family of subsets of $N$ (integral solutions)
- $\mathbf{P}_\mathcal{I}$ – a convex relaxation for the constraints imposed by $\mathcal{I}$ (fractional solutions)
  - $\mathbf{P}_\mathcal{I}$ is *down-monotone*: for $z, z' \in [0,1]^N$, if $z \leq z'$ and $z' \in \mathbf{P}_\mathcal{I}$, then $z \in \mathbf{P}_\mathcal{I}$
  - $\mathbf{P}_\mathcal{I}$ is *solvable*: one can optimize any linear function over $\mathbf{P}_\mathcal{I}$ in polynomial time

# Submodular objective: CR scheme

- $N$ – a finite ground set (tasks $\mathcal{T}$)
- $\mathcal{I} \subseteq 2^N$ – a family of subsets of $N$ (integral solutions)
- $\mathbf{P}_{\mathcal{I}}$ – a convex relaxation for the constraints imposed by $\mathcal{I}$ (fractional solutions)
  - $\mathbf{P}_{\mathcal{I}}$ is *down-monotone*: for $z, z' \in [0,1]^N$, if $z \leq z'$ and $z' \in \mathbf{P}_{\mathcal{I}}$, then $z \in \mathbf{P}_{\mathcal{I}}$
  - $\mathbf{P}_{\mathcal{I}}$ is *solvable*: one can optimize any linear function over $\mathbf{P}_{\mathcal{I}}$ in polynomial time
- $\mathbf{R}(x)$ – random sample of $N$ s.t. each $i \in N$ is in $\mathbf{R}(x)$ independently at random with probability $x_i$

# Submodular objective: CR scheme

- $N$ – a finite ground set (tasks $\mathcal{T}$)
- $\mathcal{I} \subseteq 2^N$ – a family of subsets of $N$ (integral solutions)
- $\mathbf{P}_{\mathcal{I}}$ – a convex relaxation for the constraints imposed by $\mathcal{I}$ (fractional solutions)
  - $\mathbf{P}_{\mathcal{I}}$ is *down-monotone*: for $z, z' \in [0,1]^N$, if $z \leq z'$ and $z' \in \mathbf{P}_{\mathcal{I}}$, then $z \in \mathbf{P}_{\mathcal{I}}$
  - $\mathbf{P}_{\mathcal{I}}$ is *solvable*: one can optimize any linear function over $\mathbf{P}_{\mathcal{I}}$ in polynomial time
- $\mathbf{R}(x)$ – random sample of $N$ s.t. each $i \in N$ is in $\mathbf{R}(x)$ independently at random with probability $x_i$
- For a set function $f : 2^N \to \mathbb{R}_+$ let $F : [0,1]^N \to \mathbb{R}_+$ denote the *multilinear extension* of $f$, which is defined as $F(x) := \mathbb{E}[f(\mathbf{R}(x))]$. (f – submodular objective function)

# Submodular objective: CR scheme

**Definition**

For $b, c \in [0,1]$, a $(b,c)$-balanced *CR scheme* $\pi$ for a polytope $\mathbf{P}_{\mathcal{I}}$ is a procedure that for every $x \in b \cdot \mathbf{P}_{\mathcal{I}}$ and $A \subseteq N$ returns a random set $\pi_x(A)$ satisfying

1. $\pi_x(A) \subseteq \mathrm{support}(x) \cap A$ and $\pi_x(A) \in \mathcal{I}$ with probability 1,
2. for all $i \in \mathrm{support}(x)$, $\Pr[i \in \pi_x(\mathbf{R}(x)) \mid i \in \mathbf{R}(x)] \geq c$.

Here $\mathrm{support}(x) := \{i \in N \colon x_i > 0\}$, $b \cdot \mathbf{P}_{\mathcal{I}} := \{bx \colon x \in \mathbf{P}_{\mathcal{I}}\}$.

# Submodular objective: CR scheme

## Definition

For $b, c \in [0,1]$, a $(b,c)$-balanced *CR scheme* $\pi$ for a polytope $\mathbf{P}_{\mathcal{I}}$ is a procedure that for every $x \in b \cdot \mathbf{P}_{\mathcal{I}}$ and $A \subseteq N$ returns a random set $\pi_x(A)$ satisfying

1. $\pi_x(A) \subseteq \mathrm{support}(x) \cap A$ and $\pi_x(A) \in \mathcal{I}$ with probability 1,
2. for all $i \in \mathrm{support}(x)$, $\Pr[i \in \pi_x(\mathbf{R}(x)) \mid i \in \mathbf{R}(x)] \geq c$.

Here $\mathrm{support}(x) := \{i \in N \colon x_i > 0\}$, $b \cdot \mathbf{P}_{\mathcal{I}} := \{bx \colon x \in \mathbf{P}_{\mathcal{I}}\}$.

The LP rounding algorithm for linear objective function yields a $(1/\Theta(k), 1/2)$-balanced CR scheme:

- selection phase – taking a random sample of $1/\Theta(k) \cdot x$,
- alteration phase – makes the solution feasible, each selected task is accepted with probability at least $1/2$.

# Submodular objective: CR scheme

**Theorem (Chekuri, Vondrák, Zenklusen)**

*Let $f : 2^N \to \mathbb{R}_+$ be a submodular function. Let $\mathcal{I} \subseteq 2^N$ be a family of feasible solutions and let $\mathbf{P}_\mathcal{I} \subseteq [0,1]^N$ be a convex relaxation for $\mathcal{I}$ that is down-monotone and solvable. Suppose that there is a $(b,c)$-balanced CR scheme for $\mathbf{P}_\mathcal{I}$. Then there is a polynomial time randomized algorithm that constructs a solution $I \in \mathcal{I}$ s.t.*

$$\mathbb{E}[f(I)] \geq \Theta(bc) \cdot \max\{F(x) \colon x \in \mathbf{P}_\mathcal{I}\}.$$

# Submodular objective: CR scheme

**Theorem (Chekuri, Vondrák, Zenklusen)**

*Let $f : 2^N \to \mathbb{R}_+$ be a submodular function. Let $\mathcal{I} \subseteq 2^N$ be a family of feasible solutions and let $\mathbf{P}_{\mathcal{I}} \subseteq [0,1]^N$ be a convex relaxation for $\mathcal{I}$ that is down-monotone and solvable. Suppose that there is a $(b,c)$-balanced CR scheme for $\mathbf{P}_{\mathcal{I}}$. Then there is a polynomial time randomized algorithm that constructs a solution $I \in \mathcal{I}$ s.t.*

$$\mathbb{E}[f(I)] \geq \Theta(bc) \cdot \max\{F(x) \colon x \in \mathbf{P}_{\mathcal{I}}\}.$$

For Submodular UFP-tree on intersecting instances:

- there is a $(1/\Theta(k), 1/2)$-balanced CR scheme for $\mathbf{P}_{\mathcal{I}}$, and
- $\max\{F(x) \colon x \in \mathbf{P}_{\mathcal{I}}\} = \Omega(\mathrm{OPT})$
  (obvious for poly-bounded edge capacities, as the optimal solution $\mathcal{T}^*$ is in $\mathbf{P}_{\mathcal{I}}$; for arbitrary edge capacities holds as $\frac{1}{64} \cdot \mathbf{1}_{\mathcal{T}^*} \in \mathbf{P}_{\mathcal{I}}$)

# Submodular objective: CR scheme

**Theorem**

*There is a polynomial time $O(k)$ approximation algorithm for Submodular UFP-tree on intersecting instances and, therefore, an $O(k \log n)$ approximation for arbitrary instances, where $k$ is the pathwidth of the tree.*