A $(5/3 + \epsilon)$-Approximation for Unsplittable Flow on a Path: Placing Small Tasks into Boxes

Fabrizio Grandoni[1] Tobias Mömke[2] Andreas Wiese[3] Hang Zhou[4]

[1]IDSIA, Switzerland

[2]Saarland University and University of Bremen, Germany

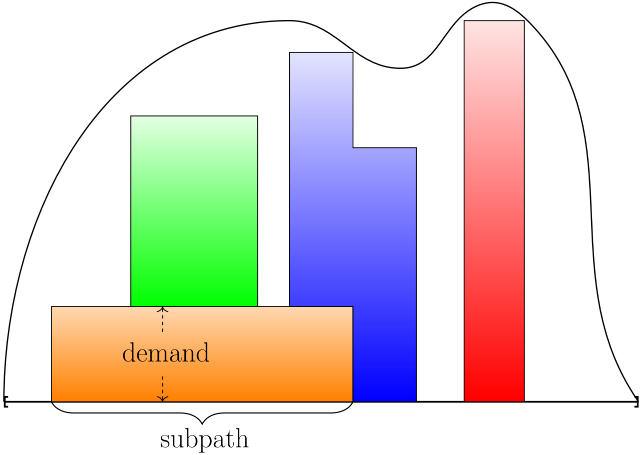
[3]University of Chile, Chile

[4]École Polytechnique, France

Symposium on Theory of Computing (STOC), 2018

Unsplittable Flow on a Path (UFP)

Task: *subpath, demand, weight*



Applications: resource allocation, caching, bandwidth allocation, scheduling, etc.

Polynomial time:

- $O(\log n)$ [Bansal, Friggstad, Khandekar, Salavatipour, SODA 2009]
- $7 + \epsilon$ [Bonsma, Schulz, Wiese, FOCS 2011]
- $2 + \epsilon$ [Anagnostopoulos, Grandoni, Leonardi, Wiese, SODA 2014]
- $1 + \epsilon$ when weight/demand is bounded [Batra, Garg, Kumar, Mömke, Wiese, SODA 2015]
- $1 + \epsilon$ when all tasks share a common edge [Grandoni, Mömke, Wiese, Zhou, SODA 2017]

Polynomial time:

- $O(\log n)$ [Bansal, Friggstad, Khandekar, Salavatipour, SODA 2009]
- $7 + \epsilon$ [Bonsma, Schulz, Wiese, FOCS 2011]
- $2 + \epsilon$ [Anagnostopoulos, Grandoni, Leonardi, Wiese, SODA 2014]
- $1 + \epsilon$ when weight/demand is bounded
  [Batra, Garg, Kumar, Mömke, Wiese, SODA 2015]
- $1 + \epsilon$ when all tasks share a common edge
  [Grandoni, Mömke, Wiese, Zhou, SODA 2017]

Polynomial time:

- $O(\log n)$ [Bansal, Friggstad, Khandekar, Salavatipour, SODA 2009]
- $7 + \epsilon$ [Bonsma, Schulz, Wiese, FOCS 2011]
- $2 + \epsilon$ [Anagnostopoulos, Grandoni, Leonardi, Wiese, SODA 2014]
- $1 + \epsilon$ when weight/demand is bounded
  [Batra, Garg, Kumar, Mömke, Wiese, SODA 2015]
- $1 + \epsilon$ when all tasks share a common edge
  [Grandoni, Mömke, Wiese, Zhou, SODA 2017]

Quasi-polynomial time:

- $1 + \epsilon$ (*) [Bansal, Chakrabarti, Epstein, Schieber, STOC 2006]
- $1 + \epsilon$ [Batra, Garg, Kumar, Mömke, Wiese, SODA 2015]

Polynomial time:

- $O(\log n)$ [Bansal, Friggstad, Khandekar, Salavatipour, SODA 2009]
- $7 + \epsilon$ [Bonsma, Schulz, Wiese, FOCS 2011]
- $2 + \epsilon$ [Anagnostopoulos, Grandoni, Leonardi, Wiese, SODA 2014]
- $1 + \epsilon$ when weight/demand is bounded
  [Batra, Garg, Kumar, Mömke, Wiese, SODA 2015]
- $1 + \epsilon$ when all tasks share a common edge
  [Grandoni, Mömke, Wiese, Zhou, SODA 2017]

Quasi-polynomial time:

- $1 + \epsilon$ (*) [Bansal, Chakrabarti, Epstein, Schieber, STOC 2006]
- $1 + \epsilon$ [Batra, Garg, Kumar, Mömke, Wiese, SODA 2015]

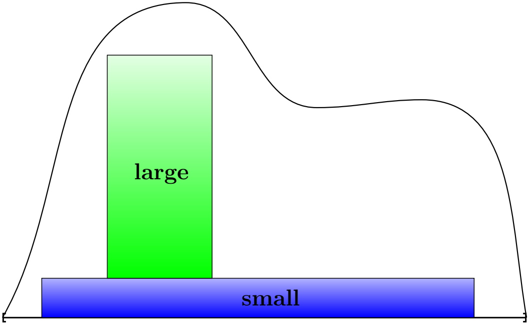
Open Question

Is there a polynomial-time approximation scheme for UFP?

Our Result

Polynomial-time $(5/3+\epsilon)$-approximation for UFP

Idea: combine large tasks and small tasks together



Previous techniques:

1. Dynamic programming: large tasks
2. Linear programming: small tasks

$$\boxed{1} + \boxed{2} \implies (2+\epsilon)\text{-approximation}$$

Q: How to achieve better-than-2 approximation?

A: **Dynamic programming with boxes:** large tasks + 1/3 of small tasks

Combined with $\boxed{2} \implies (5/3+\epsilon)$-approximation

Previous techniques:

1. Dynamic programming: large tasks
2. Linear programming: small tasks

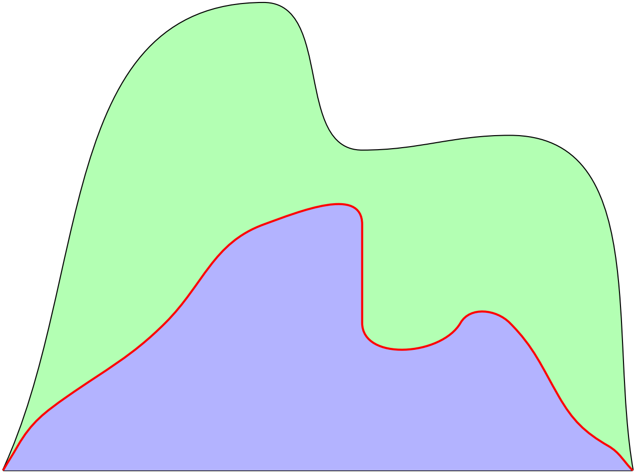
$$\boxed{1} + \boxed{2} \quad \Longrightarrow \quad (2+\epsilon)\text{-approximation}$$

Q: How to achieve better-than-2 approximation?

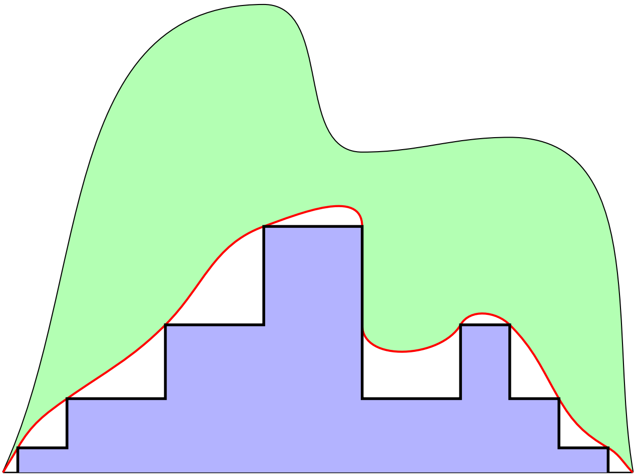
A: **Dynamic programming with boxes:** large tasks + $1/3$ of small tasks

$$\text{Combined with } \boxed{2} \quad \Longrightarrow \quad (5/3+\epsilon)\text{-approximation}$$

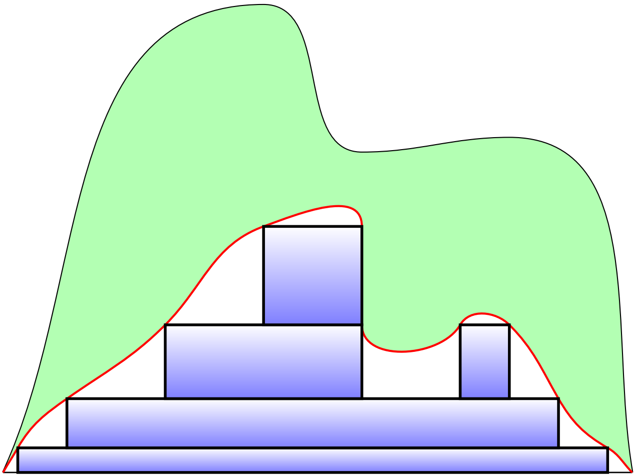
Difficulty: Unknown separation between **space for large tasks** and **space for small tasks** in the optimal solution



Preprocessing: Round down the separation profile to powers of $1 + \epsilon$

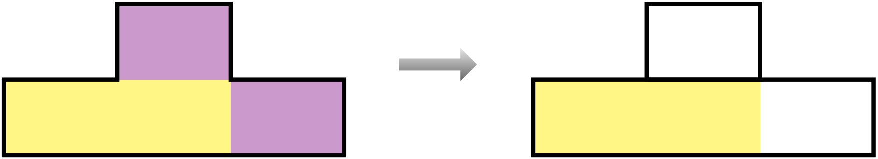


Main idea: Decompose the space for small tasks into **boxes**



Q: What factor of small tasks do we lose by introducing boxes?

A: At most 2.

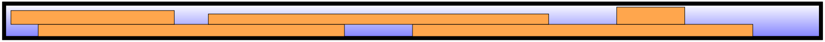


Algorithm to compute small tasks within boxes

- Guess boxes **bottom-up** using *dynamic programming*
- Fill each box with small tasks using *linear programming*

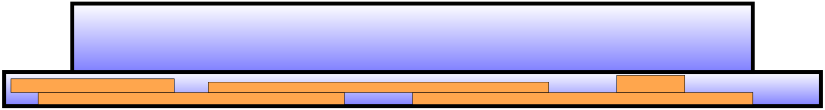
Algorithm to compute small tasks within boxes

- Guess boxes **bottom-up** using *dynamic programming*
- Fill each box with small tasks using *linear programming*



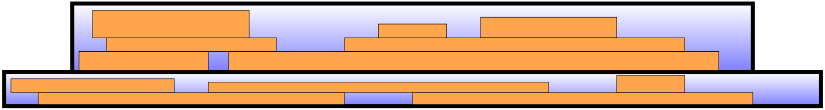
Algorithm to compute small tasks within boxes

- Guess boxes **bottom-up** using *dynamic programming*
- Fill each box with small tasks using *linear programming*



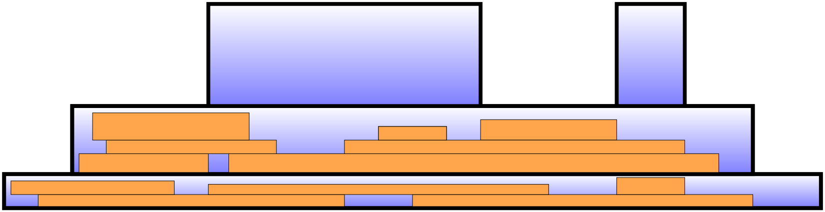
Algorithm to compute small tasks within boxes

- Guess boxes **bottom-up** using *dynamic programming*
- Fill each box with small tasks using *linear programming*



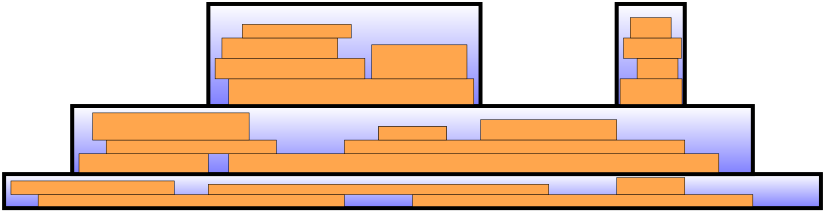
Algorithm to compute small tasks within boxes

- Guess boxes **bottom-up** using *dynamic programming*
- Fill each box with small tasks using *linear programming*



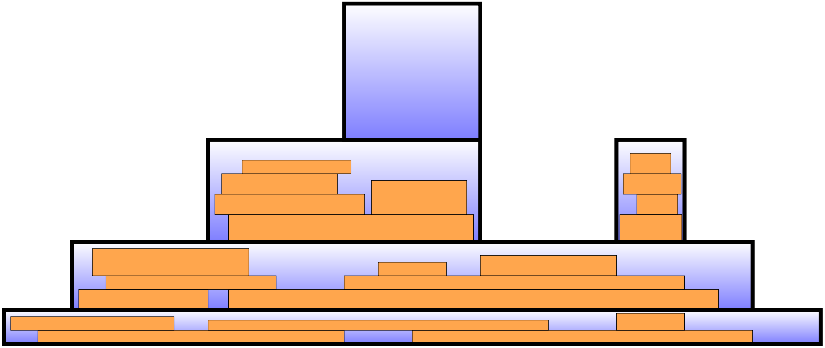
Algorithm to compute small tasks within boxes

- Guess boxes **bottom-up** using *dynamic programming*
- Fill each box with small tasks using *linear programming*



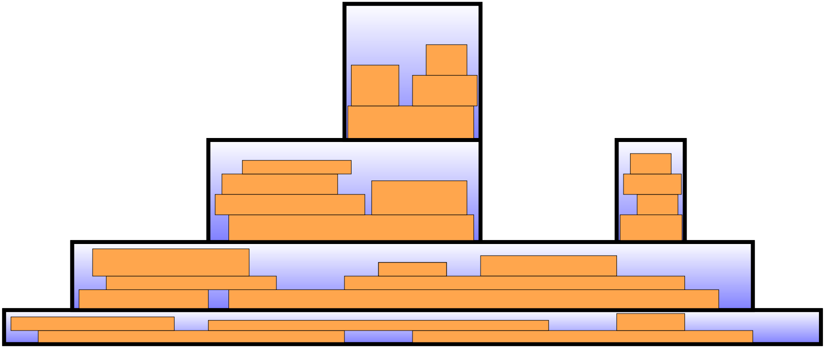
Algorithm to compute small tasks within boxes

- Guess boxes **bottom-up** using *dynamic programming*
- Fill each box with small tasks using *linear programming*

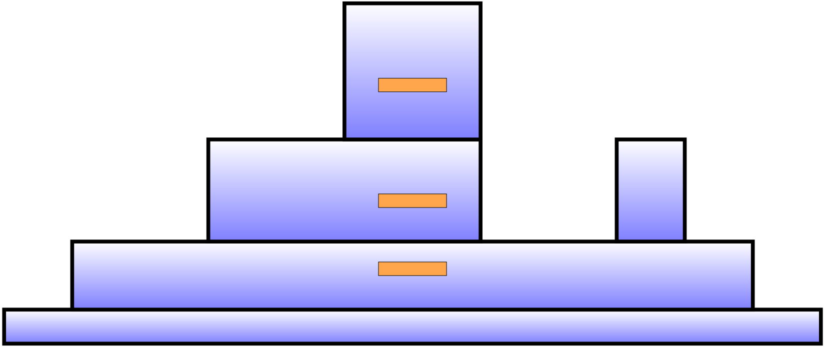


Algorithm to compute small tasks within boxes

- Guess boxes **bottom-up** using *dynamic programming*
- Fill each box with small tasks using *linear programming*

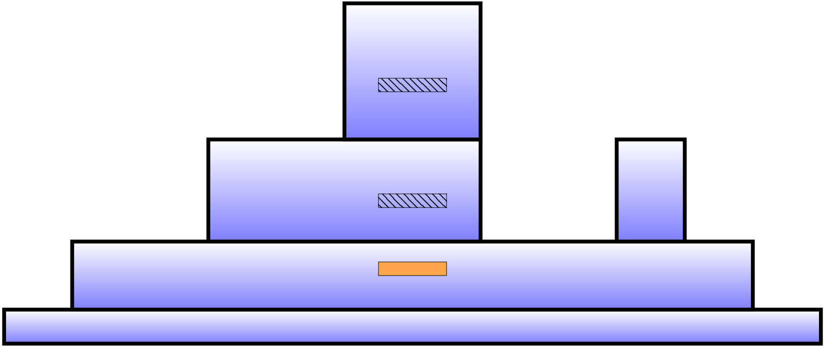


Q: How to avoid a small task being selected several times?



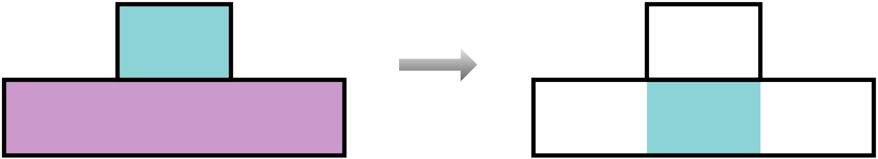
Q: How to avoid a small task being selected several times?

A: As soon as a small task is selected in some box, it is no longer allowed in upper boxes.



Q: What factor of small tasks do we lose by filling boxes bottom-up?

A: At most 2.



Loss of small tasks

- Factor 2 by introducing boxes
- Factor 2 by filling boxes bottom up

Q: Do we have to lose altogether a factor of 4 of small tasks?

Loss of small tasks

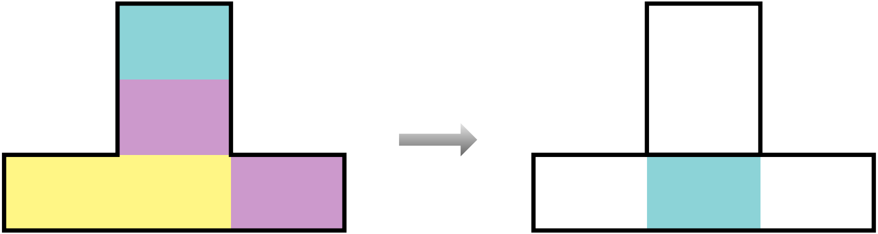
- Factor 2 by introducing boxes
- Factor 2 by filling boxes bottom up

Q: Do we have to lose altogether a factor of 4 of small tasks?

A: No. Both factors of 2 cannot happen simultaneously.

Main technical contribution

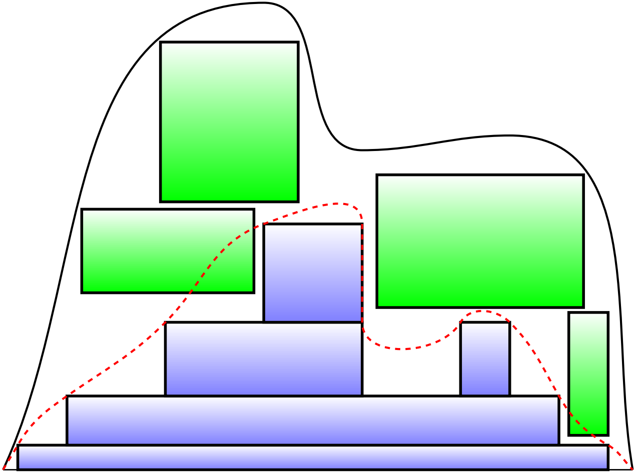
Our algorithm loses at most a factor of 3 of small tasks.



Selecting large tasks

We guess large tasks during the *dynamic program*.

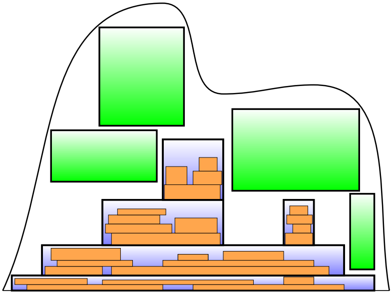
Observation: All profit from large tasks achieved when guessed correctly.



Summary

- *Dynamic programming* to guess large tasks and **boxes**
- *Linear programming* to select small tasks inside each box

Total profit: large tasks + 1/3 of small tasks



Thank you!

Our Result

Polynomial-time $(5/3+\epsilon)$-approximation for UFP