To Augment or Not to Augment: Solving Unsplittable Flow on a Path by Creating Slack

Fabrizio Grandoni  Tobias Mömke  Andreas Wiese  Hang Zhou
IDSIA, Switzerland  Saarland University  University of Chile  Max Planck Institute
Unsplittable Flow on a Path (UFP)

A task: subpath, demand, weight
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]
Polyominal 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]
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]
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]
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]

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]

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 PTAS for UFP?
Our Results

PTASes for three special cases:
- all tasks share a common edge (called rooted UFP)
- the weight of each task is propositional to its area
- a task can be included in the solution several times
Our Results

PTASes for three special cases:
- all tasks share a common edge (called *rooted UFP*)
- the weight of each task is propositional to its area
- a task can be included in the solution several times
Our Results

PTASes for three special cases:
- all tasks share a common edge (called *rooted UFP*)
- the weight of each task is propositional to its area
- a task can be included in the solution several times
Resource Augmentation

Edge capacities can be violated by an $\epsilon$-fraction.
Our framework:

PTAS with *resource augmentation* → creating slack → PTAS in general
Our framework:

PTAS with resource augmentation

creating slack

PTAS in general

In this talk:

PTAS for rooted UFP with resource augmentation

creating slack

PTAS for rooted UFP in general
PTAS for **rooted UFP** with *resource augmentation*

creating slack

PTAS for **rooted UFP** in general
$\log_{1/\epsilon} u$

- \textcolor{red}{\textbullet} type of an edge
- \textcolor{blue}{\textbullet} type of a task
$\log_{1/\epsilon} u$

- **type of an edge**
- **type of a task**
Algorithm

1. Remove the tasks of one type every $1/\epsilon$ types (shifting technique)
2. Solve each subinstance with bounded range of edge capacities
3. Output the union of the solutions to subinstances
Algorithm

1. Remove the tasks of one type every $1/\epsilon$ types (shifting technique)
2. Solve each subinstance with bounded range of edge capacities
3. Output the union of the solutions to subinstances
Algorithm

1. Remove the tasks of one type every $1/\epsilon$ types (shifting technique)
2. Solve each subinstance with bounded range of edge capacities
3. Output the union of the solutions to subinstances
Algorithm

1. Remove the tasks of one type every $1/\epsilon$ types (shifting technique)
2. Solve each subinstance with bounded range of edge capacities
3. Output the union of the solutions to subinstances
How to solve an instance with a bounded range of edge capacities?

1. observe monotonicity
2. round up edge capacities to powers of $1 + \epsilon \implies O_\epsilon(1)$ steps
3. apply PTAS for constant-dimensional knapsack [Frieze, Clarke, 1984]
How to solve an instance with a bounded range of edge capacities?

1. observe monotonicity
2. round up edge capacities to powers of $1 + \varepsilon \implies O_\varepsilon(1)$ steps
3. apply PTAS for constant-dimensional knapsack [Frieze, Clarke, 1984]
How to solve an instance with a bounded range of edge capacities?

1. observe monotonicity

2. round up edge capacities to powers of $1 + \epsilon \implies O_{\epsilon}(1)$ steps

3. apply PTAS for constant-dimensional knapsack [Frieze, Clarke, 1984]
PTAS for rooted UFP with resource augmentation

creating slack

PTAS for rooted UFP in general
New Slack Lemma

A near-optimal solution where on each edge there is some slack s.t.  
1. the number of large tasks is bounded;  
2. the slack is at least $\epsilon$ fraction of the total demand of small tasks.
New Slack Lemma

A near-optimal solution where on each edge there is some slack s.t.
1. the number of large tasks is bounded;
2. the slack is at least $\epsilon$ fraction of the total demand of small tasks.

Question: How to obtain Property 2?
Answer: Shrink $\epsilon$ fraction of small tasks.

Small integrality gap
[Chekuri, Mydlarz, Shepherd, 2007]
New Slack Lemma

A near-optimal solution where on each edge there is some slack s.t.
1. the number of large tasks is bounded;
2. the slack is at least $\epsilon$ fraction of the total demand of small tasks.

Question: Why does Property 2 help?

Answer: Borrow techniques from the resource augmentation setting.
PTAS for rooted UFP with resource augmentation creating slack

PTAS for rooted UFP in general
Our dynamic program proceeds in increasing order of types. For each type, it guesses:

- large tasks
- capacity profile for small tasks

\{ \text{constant complexity} \}
**Difficulty:** Not able to remember previously guessed information.

**Solution:** Remember information only from the last type.
Difficulty: Not able to remember previously guessed information.
Solution: Remember information only from the last type.
**Difficulty:** Not able to remember previously guessed information.

**Solution:** Remember information only from the last type.

small tasks of type $j$
Difficulty: Not able to remember previously guessed information.

Solution: Remember information only from the last type.
Difficulty: Not able to remember previously guessed information.
Solution: Remember information only from the last type.
Difficulty: Not able to remember previously guessed information.
Solution: Remember information only from the last type.

small tasks of type $j$

small tasks of type $j - 2$

forgotten information

slack at type $j$ accommodates the forgotten information
Difficulty: Not able to remember previously guessed information.
Solution: Remember information only from the last type.

Polynomial-time dynamic program
Conclusion

In this talk:

PTAS for **rooted UFP** with *resource augmentation*

creating slack

PTAS for **rooted UFP** in general
Conclusion

Our framework:

PTAS with \textit{resource augmentation}

creating slack

PTAS in general
Conclusion

Our framework:

PTAS with resource augmentation

creating slack

PTAS in general

Open Question

Is there a PTAS for general UFP (even with resource augmentation)?
Thank you!