A practical FPT algorithm for Flow Decomposition and transcript assembly

---

Kyle Kloster, **Philipp Kuinke**, Michael P. O'Brien, Felix Reidl, Fernando Sánchez Villaamil, Blair D. Sullivan, Andrew van der Poel

2017/07/19

North Carolina State University
RWTH Aachen University

Motivation

The Problem

AGGACGTAGATAGCTAGCTAATGCTACGATCAGAGGACGTAGATTTATTACCAT

TACCGAATACGAACTAGGATATCGATCGATCAGAGGCCCAATAGGGAATATCCG

TACCGAATACGAACTAGGATATCGATCGATTGATCTATAATAGTAGAATATCCG

The Problem

AGGACGTAG ATAGCTAGC ATATCGATC GATCAGAGG ACGTAGATT TATTACCAT

TACCGAATA CGAACTAGG ATATCGATC GATCAGAGG CCCAATAGG GAATATCCG

TACCGAATA CGAACTAGG ATATCGATC GATTGATCT ATAATAGTA GAATATCCG

The Problem

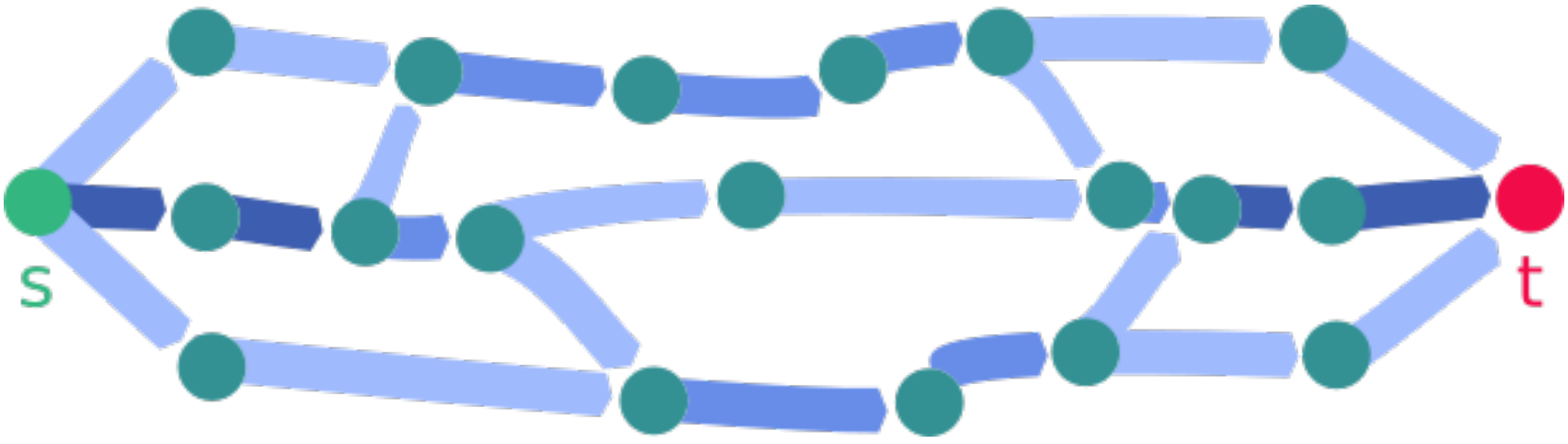
AGGACGTAG	ATAGCTAGC	TAATGCTAC	GATCAGAGG	ACGTAGATT	TATTACCAT
TACCGAATA	CGAACTAGG	ATATCGATC	GATCAGAGG	CCCAATAGG	GAATATCCG
TACCGAATA	CGAACTAGG	ATATCGATC	GATTGATCT	ATAATAGTA	GAATATCCG

Shared segments between DNA/RNA strands create ambiguity in the assembly problem

The Problem

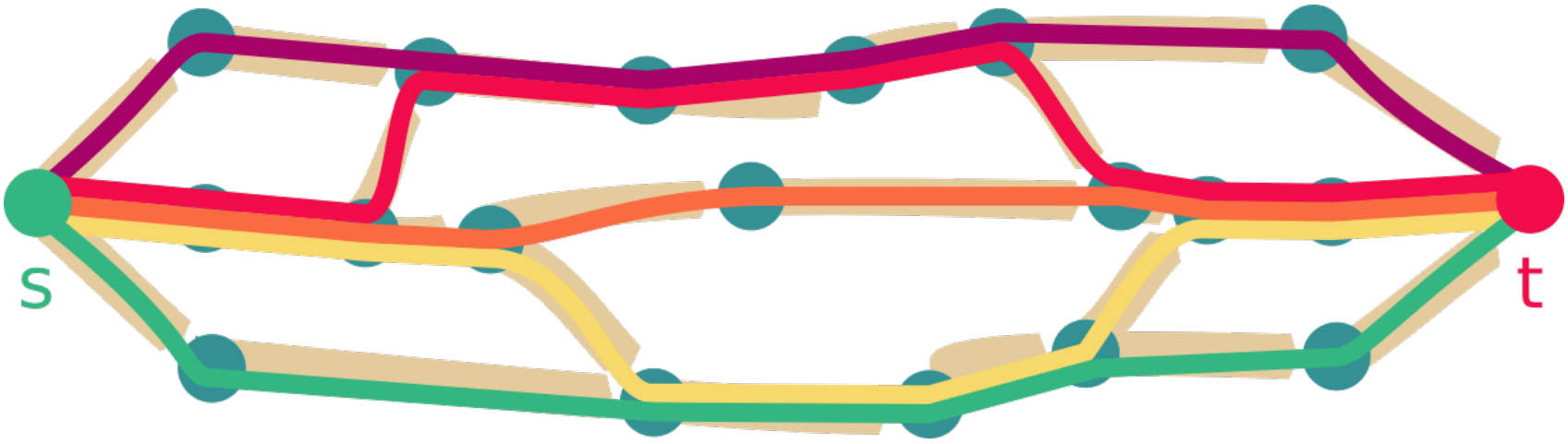
Shared segments between DNA/RNA strands create ambiguity in the assembly problem

The Problem



Connecting overlapping segments and counting their frequencies yields a DAG and a flow

The Problem



The problem is to split the flow int the least amount of s-t-paths, to recover the original DNA/RNA strands.

The Problem

k-Flow Decomposition (k-FD)

Input: (G, f, k) with G an s-t–DAG, f a flow on G, and k a positive integer.

Problem: Is there an integral *flow decomposition* of (G, f) using at most k paths?

The Problem

k-Flow Decomposition (*k*-FD)

Input: (G, f, k) with G an s-t–DAG, f a flow on G, and k a positive integer.

Problem: Is there an integral *flow decomposition* of (G, f) using at most k paths?

The Problem

k-Flow Decomposition (k-FD)

Input: (G, f, k) with G an s-t–DAG, f a flow on G, and k a positive integer.

Problem: Is there an integral *flow decomposition* of (G, f) using at most k paths?

- Set of s-t–paths $P = p_1, \ldots, p_k$

The Problem

k-Flow Decomposition (k-FD)

Input: (G, f, k) with G an s-t–DAG, f a flow on G, and k a positive integer.

Problem: Is there an integral *flow decomposition* of (G, f) using at most k paths?

- Set of s-t–paths $P = p_1, \ldots, p_k$
- Weights $\mathbf{w} = (w_1, \ldots, w_k)$

The Problem

k-Flow Decomposition (k-FD)

Input: (G, f, k) with G an s-t–DAG, f a flow on G, and k a positive integer.

Problem: Is there an integral *flow decomposition* of (G, f) using at most k paths?

- Set of s-t–paths $P = p_1, \ldots, p_k$
- Weights $\mathbf{w} = (w_1, \ldots, w_k)$

$$f(a) = \sum_{i=1}^{k} w_i \cdot [a \in p_i] \quad \forall a \in A(G)$$

The Problem

k-Flow Decomposition (*k*-FD)

Input: (G, f, k) with G an s-t–DAG, f a flow on G, and k a positive integer.

Problem: Is there an integral *flow decomposition* of (G, f) using at most k paths?

- Set of s-t–paths $P = p_1, \ldots, p_k$
- Weights $\mathbf{w} = (w_1, \ldots, w_k)$

$$f(a) = \sum_{i=1}^{k} w_i \cdot [a \in p_i] \quad \forall a \in A(G)$$

Problem is NP-hard even for weights $\{1, 2, 4\}$

Computer Scientists…

About ten years ago, some computer scientists came by and said they heard we have some really cool problems. They showed that the problems are NP-complete and went away!

-Joseph Felsenstein (Biologist)

Computer Scientists…

About ten years ago, some computer scientists came by and said they heard we have some really cool problems. They showed that the problems are NP-complete and went away!

-Joseph Felsenstein (Biologist)

FPT

Definition

A parameterized problem L is *linear fixed-parameter tractable* if the question "$(x, k) \in L$?" can be decided in $f(k)\, O(|x|)$.

Observations

Investigation of the setup and data used by Shao and Kingsford:

Observations

Investigation of the setup and data used by Shao and Kingsford:

1. 99% of instances decomposition into ≤ 8 paths.

Observations

Investigation of the setup and data used by Shao and Kingsford:

1. 99% of instances decomposition into ≤ 8 paths.
   $\rightarrow$ exploit **small natural parameter**.

Observations

Investigation of the setup and data used by Shao and Kingsford:

1. 99% of instances decomposition into ≤ 8 paths.
   $\rightarrow$ exploit **small natural parameter**.
2. Data set contains ~4 million mostly small instances.

Observations

Investigation of the setup and data used by Shao and Kingsford:

1. 99% of instances decomposition into ≤ 8 paths.
   $\rightarrow$ exploit **small natural parameter**.
2. Data set contains ~4 million mostly small instances.
   $\rightarrow$ handle **large throughput**.

Observations

Investigation of the setup and data used by Shao and Kingsford:

1. 99% of instances decomposition into ≤ 8 paths.
   $\rightarrow$ exploit **small natural parameter**.
2. Data set contains ~4 million mostly small instances.
   $\rightarrow$ handle **large throughput**.
3. Output decompositions (not only decision problem).

Observations

Investigation of the setup and data used by Shao and Kingsford:

1. 99% of instances decomposition into ≤ 8 paths.
   $\rightarrow$ exploit **small natural parameter**.
2. Data set contains ~4 million mostly small instances.
   $\rightarrow$ handle **large throughput**.
3. Output decompositions (not only decision problem).
   $\rightarrow$ reliably recover **domain-specific solution**.

Toboggan

Theorem

There is an $2^{O(k^2)}(n + \lambda)$ *algorithm for solving* k-FD*, where* λ *is the logarithm of the largest flow value.*

Toboggan

Theorem

There is an $2^{O(k^2)}(n + \lambda)$ *algorithm for solving* k-FD*, where* λ *is the logarithm of the largest flow value.*

- Run-time competitive with current state of the art heuristic

Toboggan

Theorem

There is an $2^{O(k^2)}(n + \lambda)$ *algorithm for solving* k-FD, *where* λ *is the logarithm of the largest flow value.*

- Run-time competitive with current state of the art heuristic
- Worst-case run-time is linear in n

Toboggan

Theorem

There is an $2^{O(k^2)}(n + \lambda)$ *algorithm for solving* k-FD, *where* λ *is the logarithm of the largest flow value.*

- Run-time competitive with current state of the art heuristic
- Worst-case run-time is linear in n
- Guarantees optimal solution

Toboggan

Theorem

There is an $2^{O(k^2)}(n + \lambda)$ *algorithm for solving* k-FD, *where* λ *is the logarithm of the largest flow value.*

- Run-time competitive with current state of the art heuristic
- Worst-case run-time is linear in n
- Guarantees optimal solution
- Usable in practice

Toboggan

Theorem

There is an $2^{O(k^2)}(n + \lambda)$ *algorithm for solving* k-FD, *where* λ *is the logarithm of the largest flow value.*

- Run-time competitive with current state of the art heuristic
- Worst-case run-time is linear in n
- Guarantees optimal solution
- Usable in practice (*so we claim*)

Preliminaries

Related Work

- greedy-width (length)

Related Work

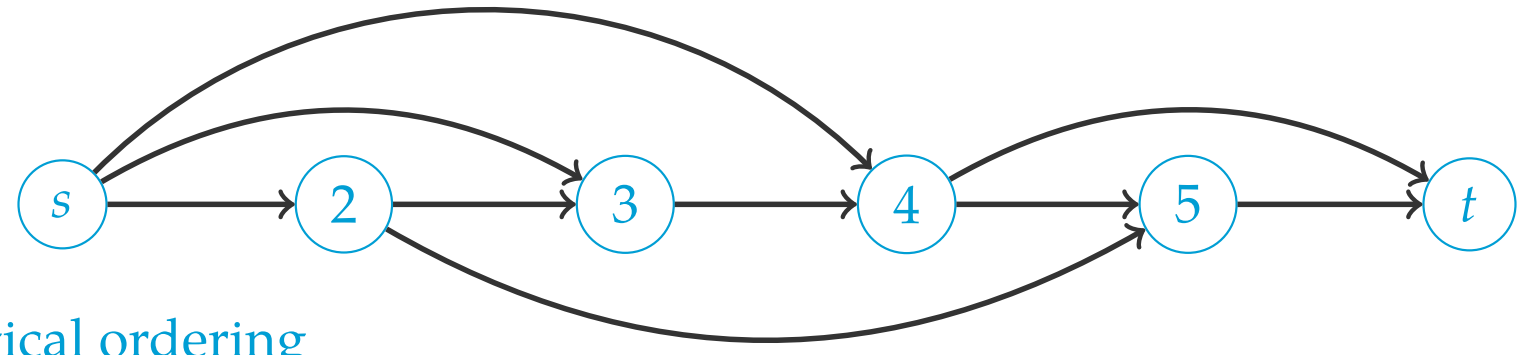
- greedy-width (length)
- Catfish

Related Work

- greedy-width (length)
- Catfish

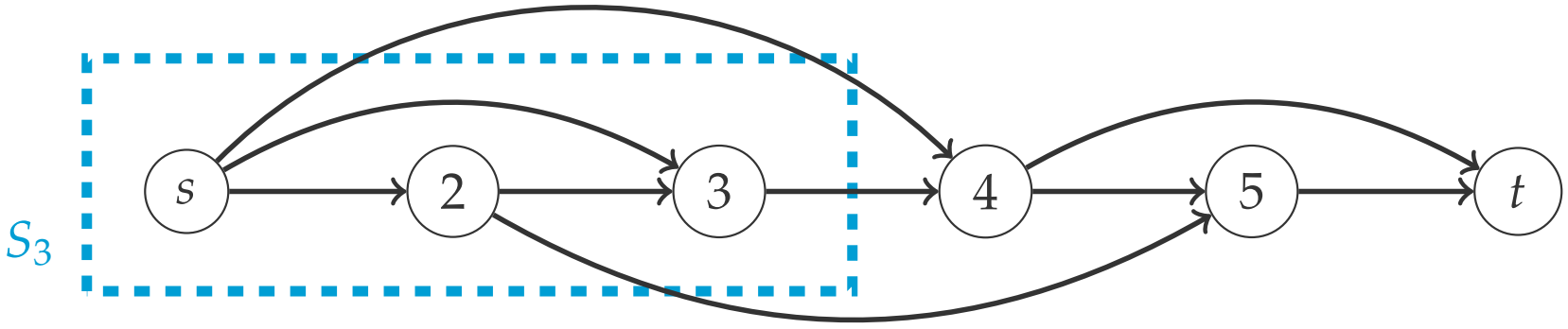
So far no exact algorithm

Definitions

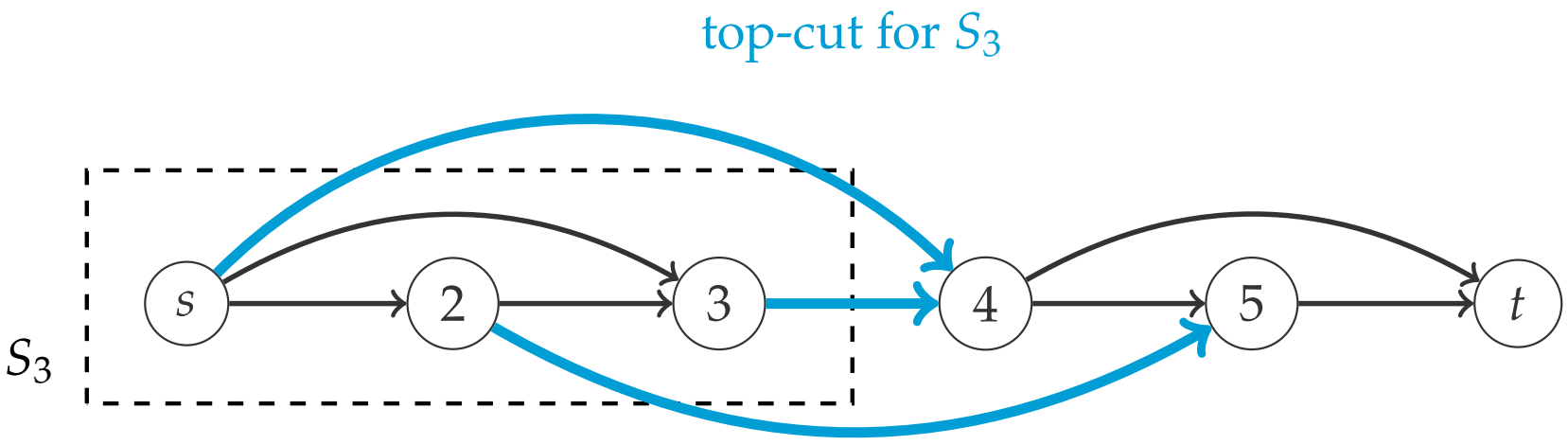


topological ordering

Definitions

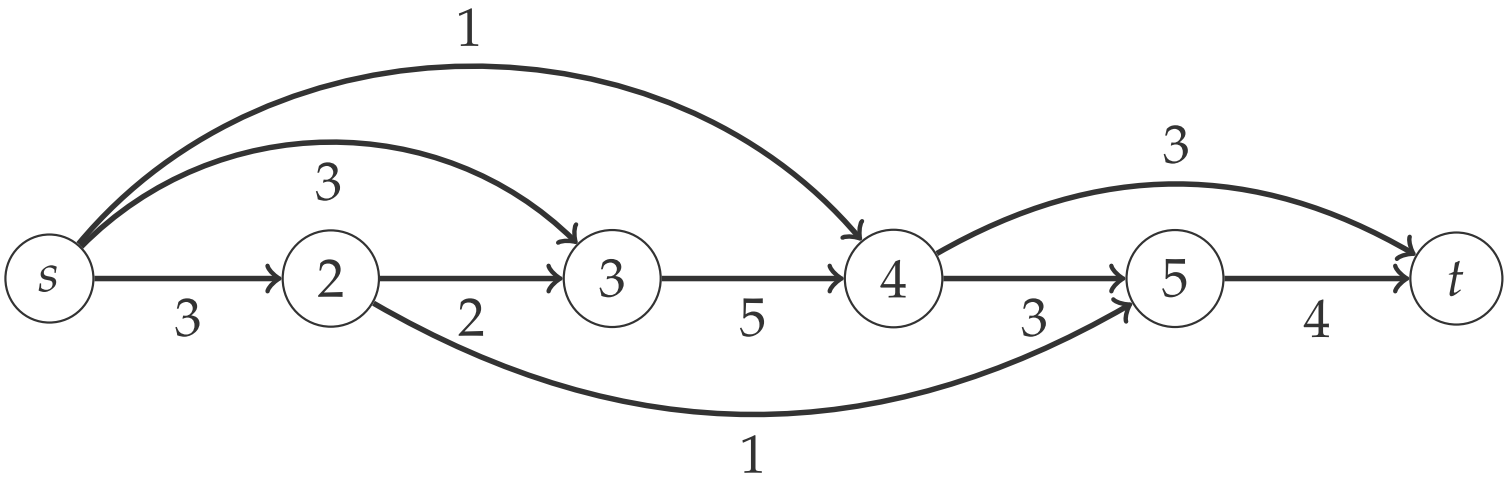


Definitions

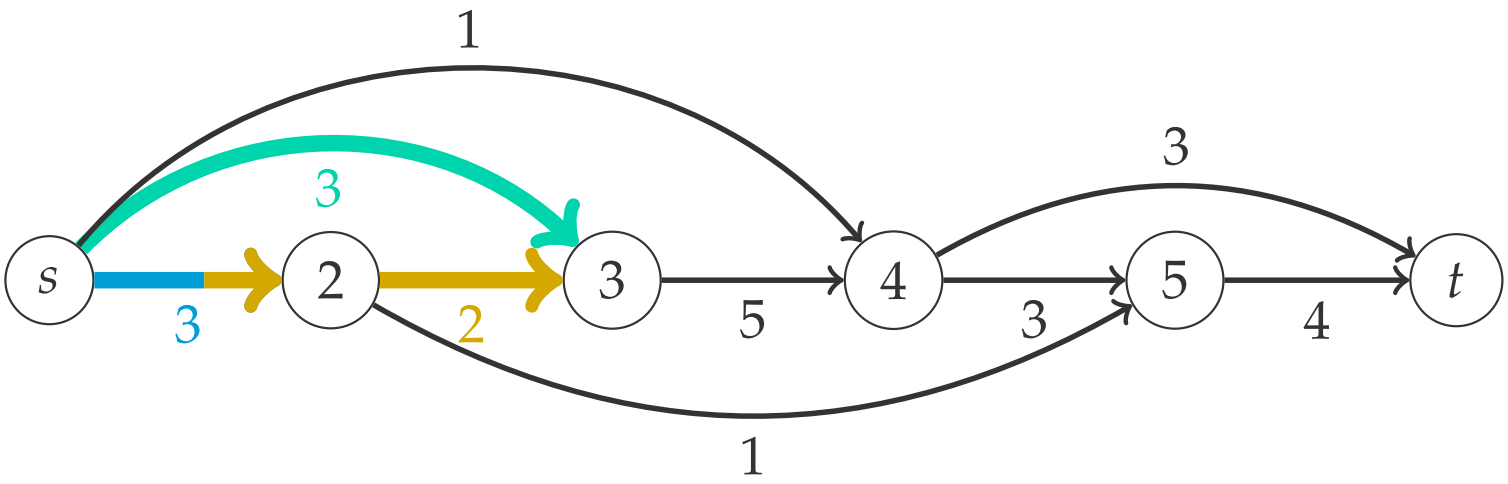


Algorithm

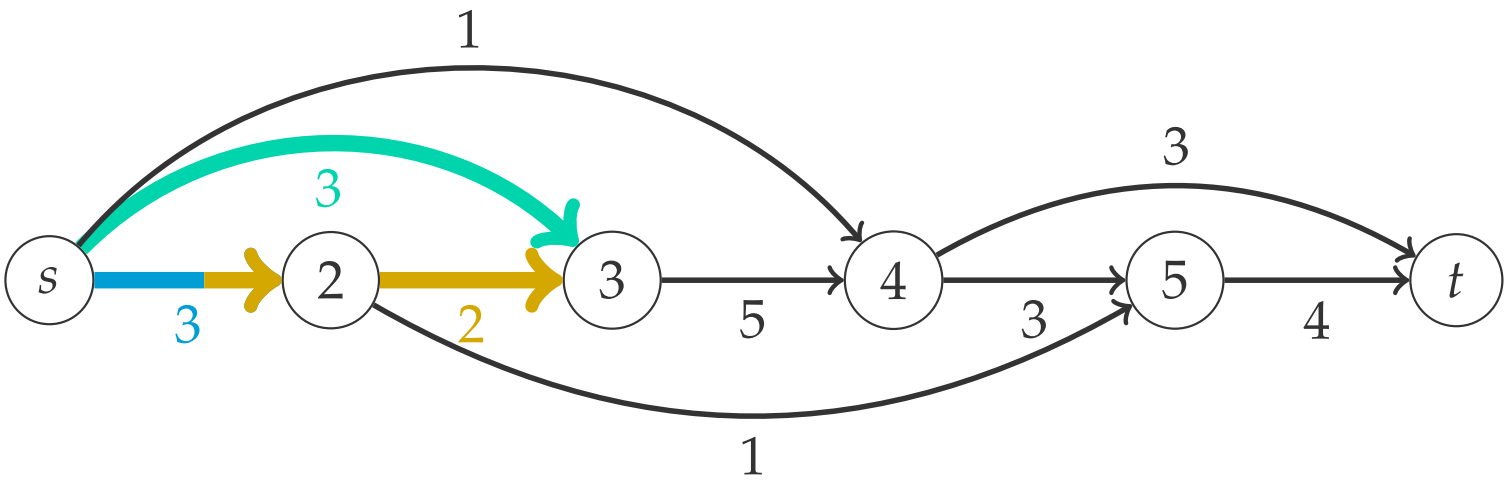
The Idea



The Idea



The Idea

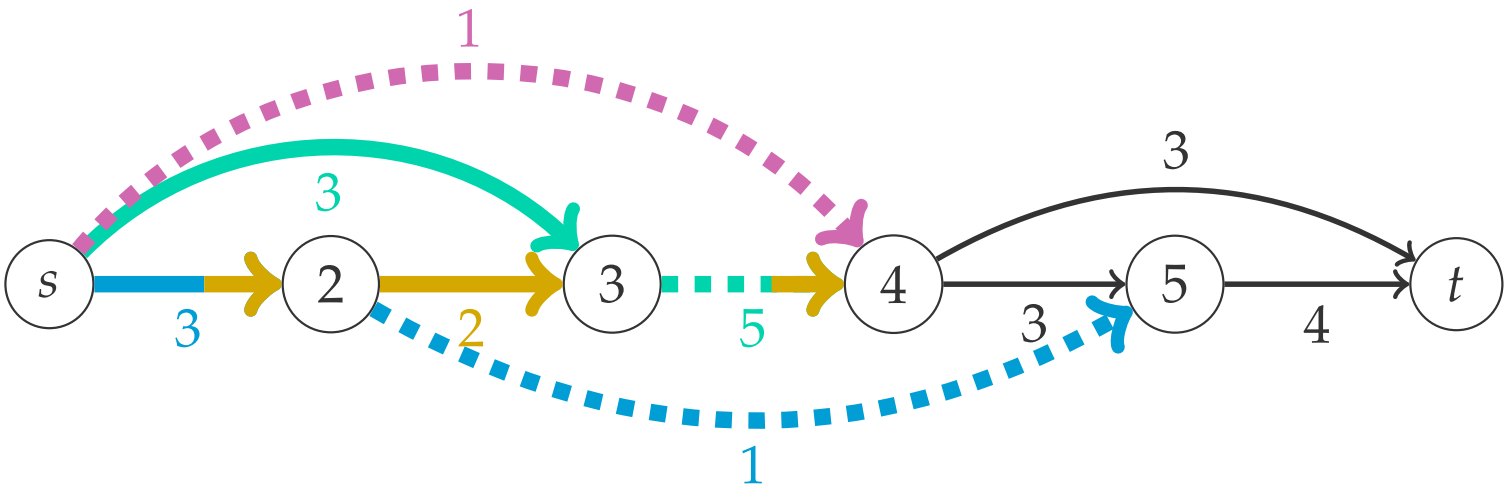


$$w_{\blacksquare} + w_{\blacksquare} = 3$$

$$w_{\blacksquare} = 3$$

$$w_{\blacksquare} = 2$$

The Idea

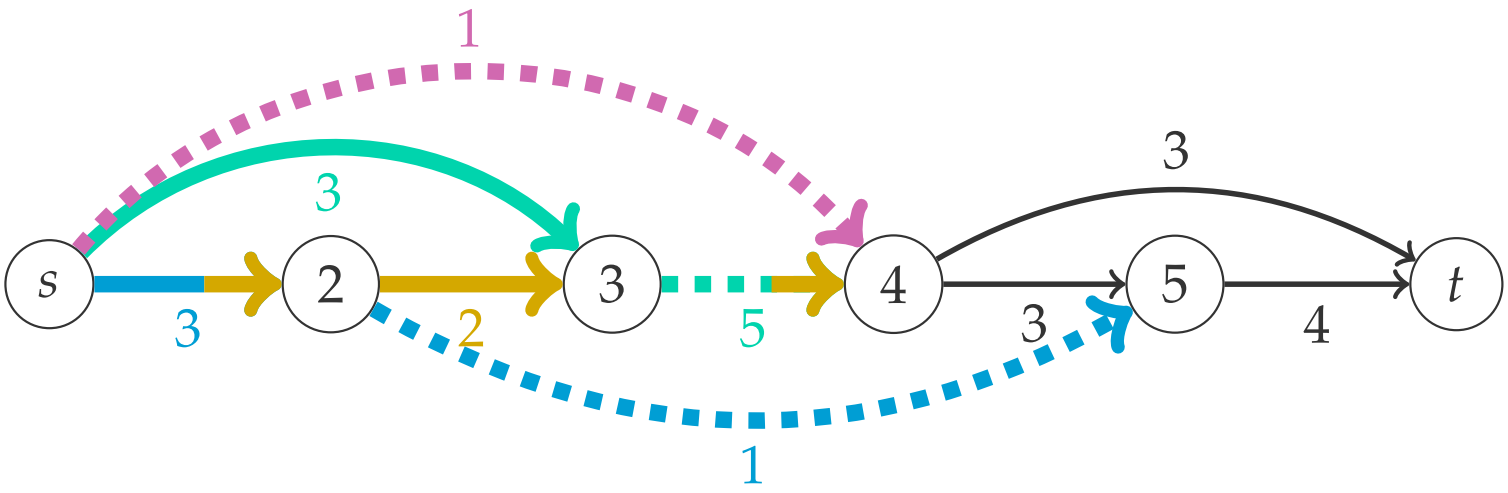


$$w_{\blacksquare} + w_{\blacksquare} = 3$$

$$w_{\blacksquare} = 3$$

$$w_{\blacksquare} = 2$$

The Idea



$$w_{\blacksquare} + w_{\blacksquare} = 3$$

$$w_{\blacksquare} = 3$$

$$w_{\blacksquare} = 2$$

$$w_{\blacksquare} = 1$$

$$w_{\blacksquare} + w_{\blacksquare} = 5$$

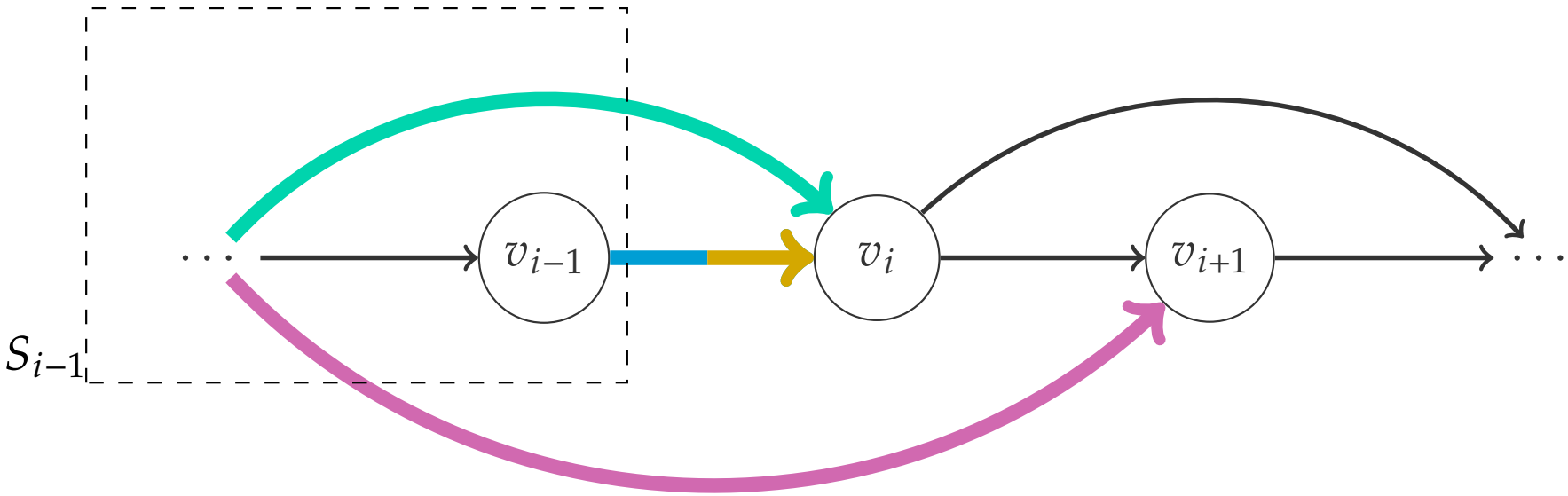
$$w_{\blacksquare} = 1$$

Dynamic Programming Step

From S_i to S_{i+1} exactly one vertex (v_{i+1}) becomes internal. We need to push the incoming flows of v_i to the outgoing edges.

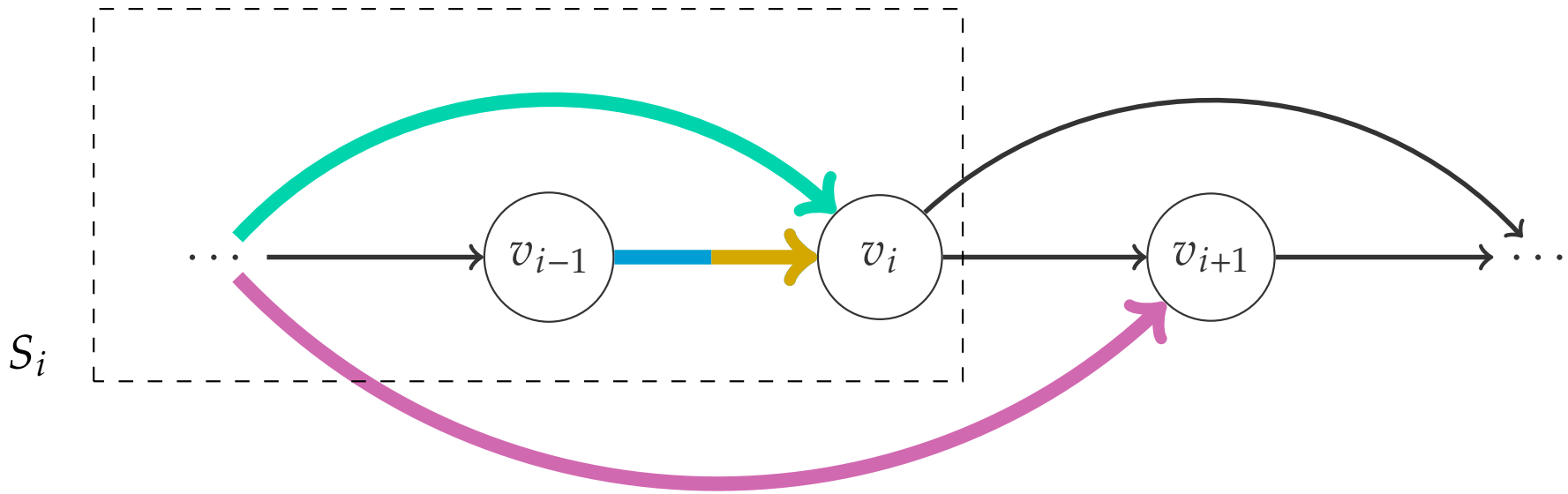
Dynamic Programming Step

From S_i to S_{i+1} exactly one vertex (v_{i+1}) becomes internal. We need to push the incoming flows of v_i to the outgoing edges.



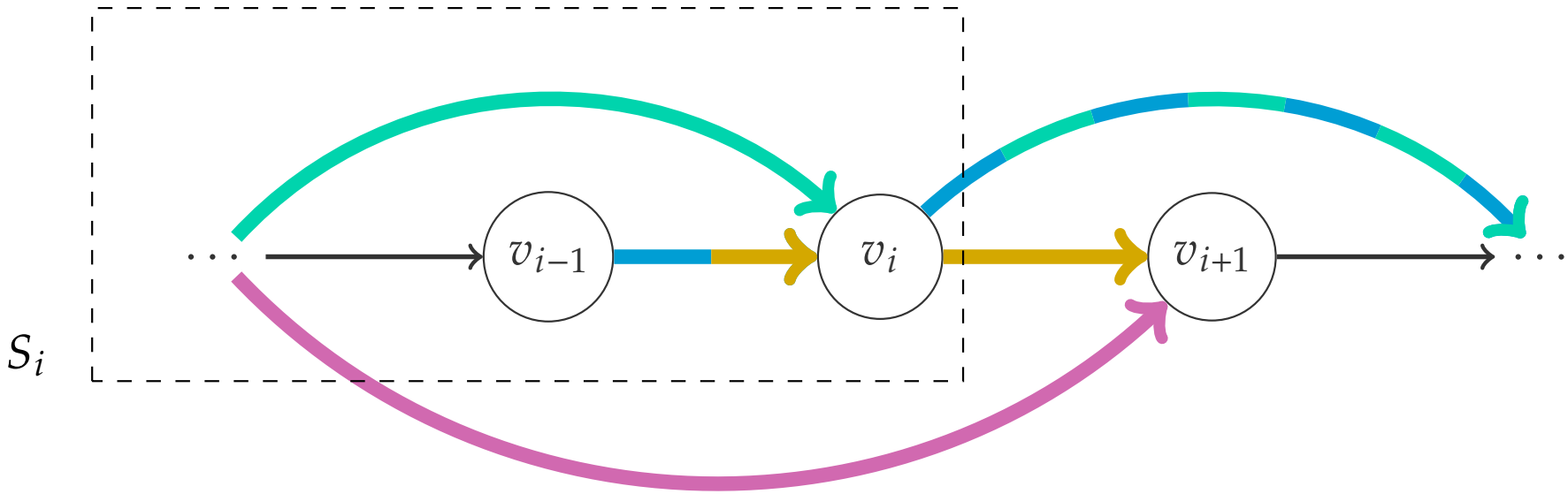
Dynamic Programming Step

From S_i to S_{i+1} exactly one vertex (v_{i+1}) becomes internal. We need to push the incoming flows of v_i to the outgoing edges.



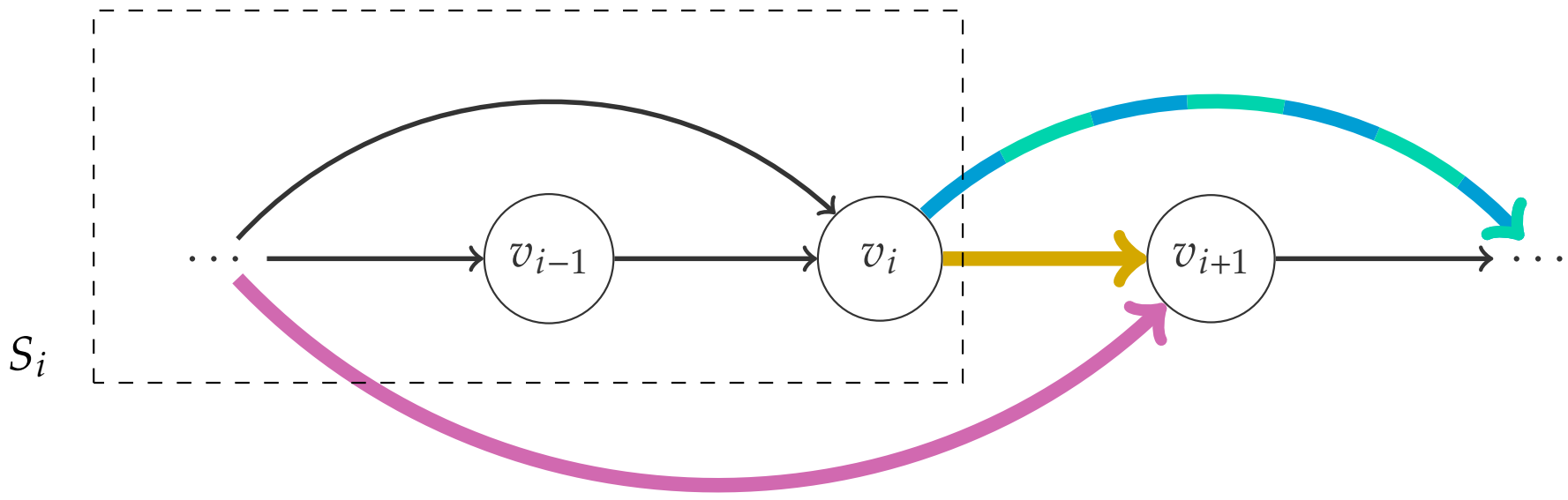
Dynamic Programming Step

From S_i to S_{i+1} exactly one vertex (v_{i+1}) becomes internal. We need to push the incoming flows of v_i to the outgoing edges.



Dynamic Programming Step

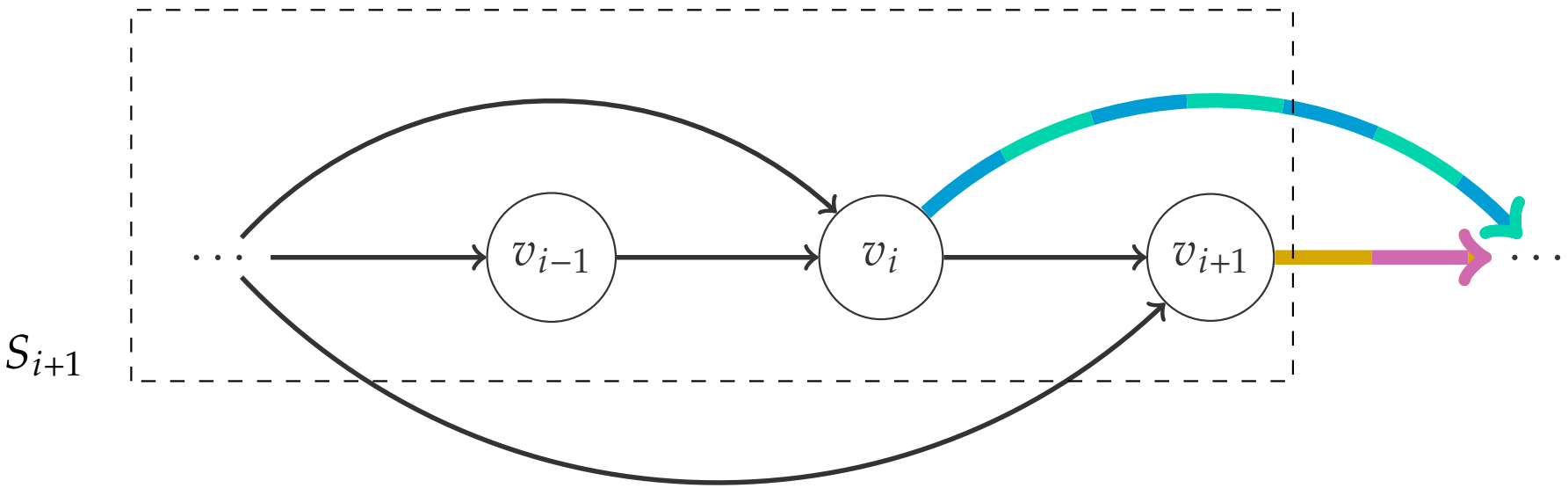
From S_i to S_{i+1} exactly one vertex (v_{i+1}) becomes internal. We need to push the incoming flows of v_i to the outgoing edges.



We can "forget" internal labels.

Dynamic Programming Step

From S_i to S_{i+1} exactly one vertex (v_{i+1}) becomes internal. We need to push the incoming flows of v_i to the outgoing edges.



Dynamic Programming

- At each step we have to label the new top-cut, which gives new constraints.

Dynamic Programming

- At each step we have to label the new top-cut, which gives new constraints.
- Whenever there are multiple possibilities we add all of them to memory.

Dynamic Programming

- At each step we have to label the new top-cut, which gives new constraints.
- Whenever there are multiple possibilities we add all of them to memory.
- Each such memory entry consists of linear system and routing for the top-cut.

Dynamic Programming

- At each step we have to label the new top-cut, which gives new constraints.
- Whenever there are multiple possibilities we add all of them to memory.
- Each such memory entry consists of linear system and routing for the top-cut.
- Entries with inconsistent linear systems will be deleted, so in practice memory consumption is much less than the theoretical maximum.

Dynamic Programming

- At each step we have to label the new top-cut, which gives new constraints.
- Whenever there are multiple possibilities we add all of them to memory.
- Each such memory entry consists of linear system and routing for the top-cut.
- Entries with inconsistent linear systems will be deleted, so in practice memory consumption is much less than the theoretical maximum.
- When we arrive at S_n we check each linear system for valid solution.

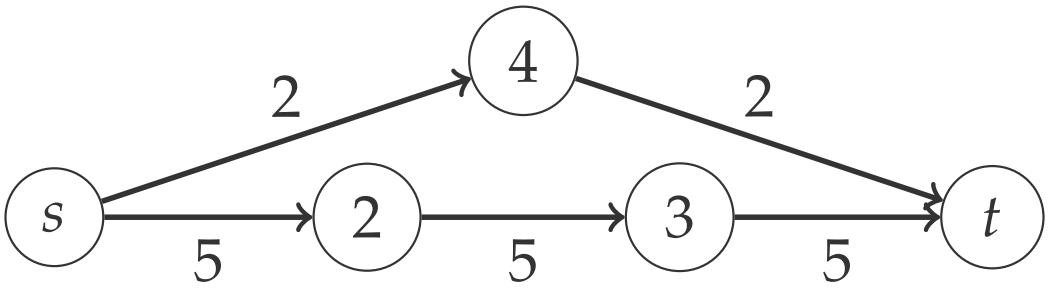
Always true?

Conjecture

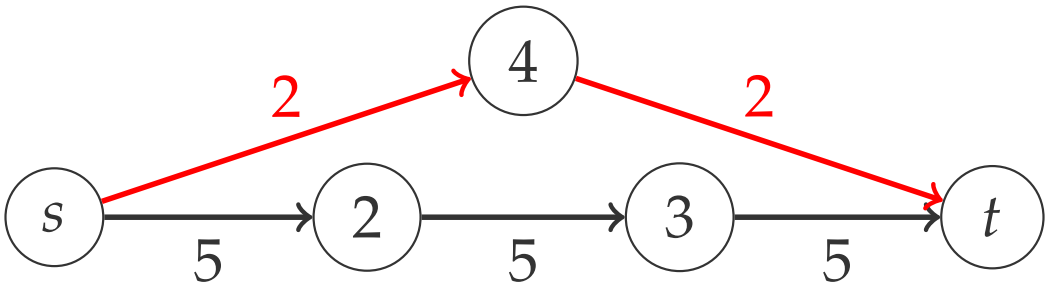
If k is the minimum value for which (G, f) has a flow decomposition of size k, then every integer-weighted solution has a corresponding linear system L of rank k.

Implementation

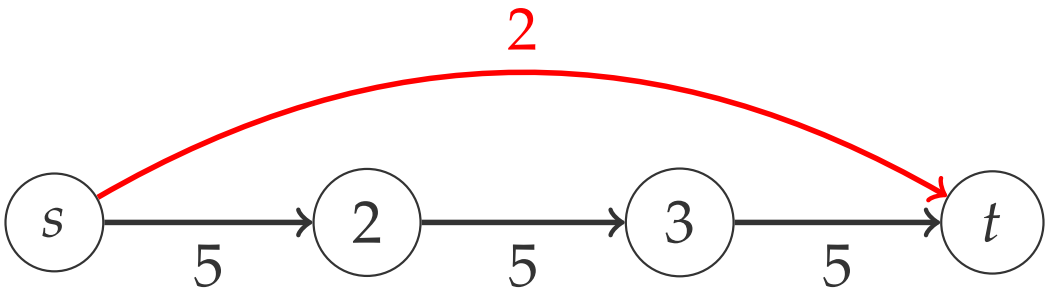
Graph Reduction



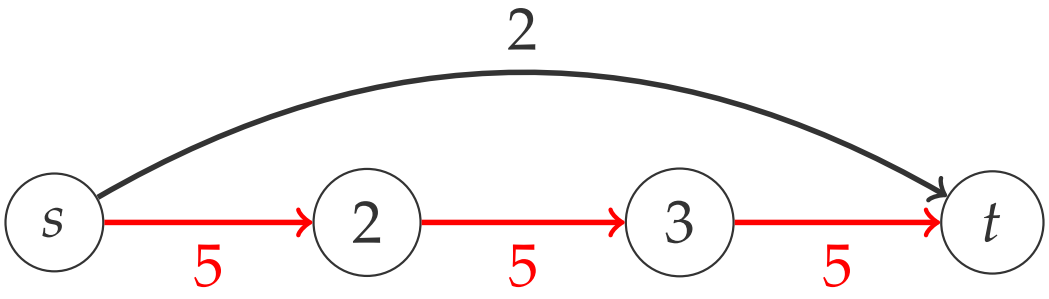
Graph Reduction



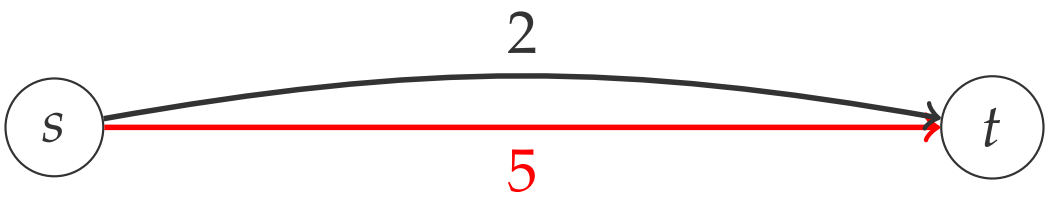
Graph Reduction



Graph Reduction



Graph Reduction



bounds on k and w

Finding k: search on $[0, b]$. How to choose b?

bounds on k and w

Finding k: search on $[0, b]$. How to choose b?

First choice: size of the largest top-cut.

bounds on k and w

Finding k: search on $[0, b]$. How to choose b?

First choice: size of the largest top-cut.

Second choice: bound given by technical lemma.

bounds on k and w

Finding k: search on $[0, b]$. How to choose b?

First choice: size of the largest top-cut.

Second choice: bound given by technical lemma.

weight vector: Restrict to values of f.

bounds on k and w

Finding k: search on $[0, b]$. How to choose b?

First choice: size of the largest top-cut.

Second choice: bound given by technical lemma.

weight vector: Restrict to values of f.

Reduce fixed weights when no solution is found.

bounds on k and w

Finding k: search on $[0, b]$. How to choose b?

First choice: size of the largest top-cut.

Second choice: bound given by technical lemma.

weight vector: Restrict to values of f.

Reduce fixed weights when no solution is found.

Additional pruning is performed according to heuristics.

Repository

Python implementation is available on Github:

`https://github.com/theoryinpractice/toboggan`

Experiments

Setup

Dataset: Available from Shao and Kingsford.

Simulated RNA sequencing data for `human`, `mouse` and `zebrafish`, containing ground-truth.

Setup

Dataset: Available from Shao and Kingsford.

Simulated RNA sequencing data for `human`, `mouse` and `zebrafish`, containing ground-truth.

Deviation from original setup:
trivial instances omitted. removes around 64% of the 4M graphs.

Setup

Dataset: Available from Shao and Kingsford.

Simulated RNA sequencing data for `human`, `mouse` and `zebrafish`, containing ground-truth.

Deviation from original setup:
trivial instances omitted. removes around 64% of the 4M graphs.

All experiments on a dedicated system with Intel i7-3770: 3.40 GHz, 8 cores, 8192 KB cache and 32 GB RAM.

Setup

Dataset: Available from Shao and Kingsford.

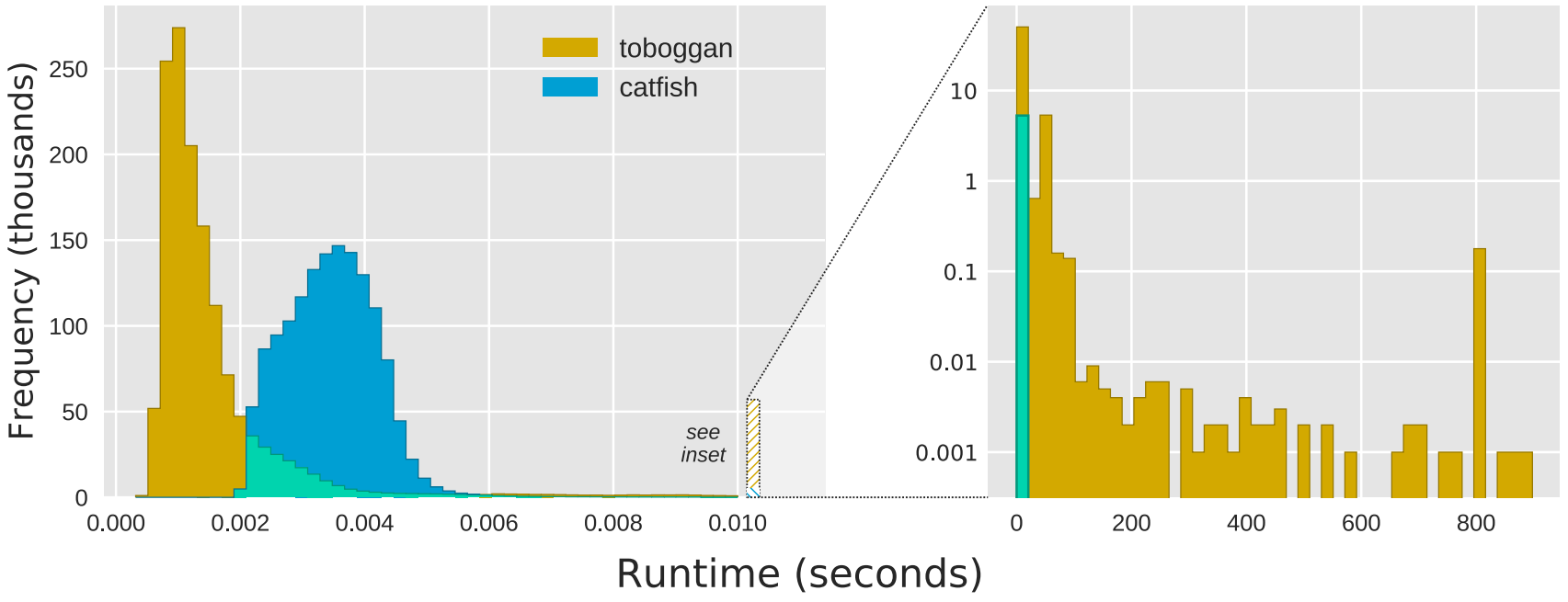
Simulated RNA sequencing data for `human`, `mouse` and `zebrafish`, containing ground-truth.

Deviation from original setup:
trivial instances omitted. removes around 64% of the 4M graphs.

All experiments on a dedicated system with Intel i7-3770: 3.40 GHz, 8 cores, 8192 KB cache and 32 GB RAM.

Toboggan had a 800 second timeout limit.

Execution Time



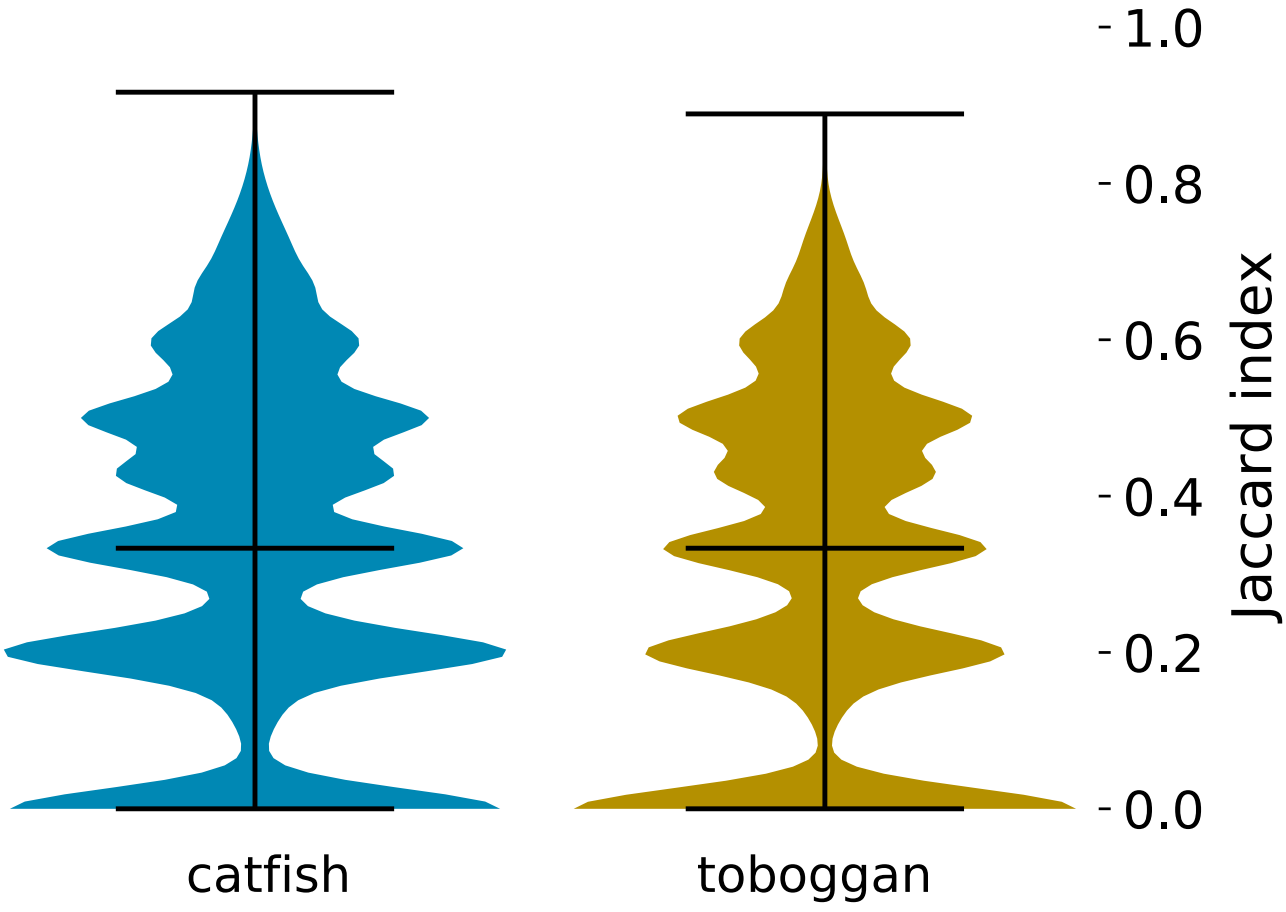
Ground Truth Validation

dataset	instances	non-trivial	optimal	non-optimal
`zebrafish`	1,549,373	445,880	99.907%	0.053%
`mouse`	1,316,058	473,185	99.401%	0.074%
`human`	1,169,083	529,523	99.490%	0.043%
all	4,034,514	1,448,588	99.589%	0.056%

Exact Recovery

k	instances	Catfish	Toboggan
2	63.2791%	0.992	**0.995**
3	22.0775%	0.967	**0.969**
4	8.5237%	**0.931**	0.930
5	3.4920%	0.886	0.886
6	1.5375%	**0.830**	0.828
7	0.6698%	**0.788**	0.780
8	0.2889%	**0.767**	0.766
9	0.1241%	0.740	**0.743**
10	0.0070%	0.752	**0.802**
11	0.0004%	0.500	0.500
all	100%	0.973	**0.975**

Solutions vs. Ground Truth



Conclusion

Conclusion

- Theoretical runtime linear in n

Conclusion

- Theoretical runtime linear in n
- competitive runtime with heuristics in practice

Conclusion

- Theoretical runtime linear in n
- competitive runtime with heuristics in practice
- guarantees optimal k

Conclusion

- Theoretical runtime linear in n
- competitive runtime with heuristics in practice
- guarantees optimal k
- even faster for efficiency first implementation (C++)?

Conclusion

- Theoretical runtime linear in n
- competitive runtime with heuristics in practice
- guarantees optimal k
- even faster for efficiency first implementation (C++)?

paper: `https://arxiv.org/abs/1706.07851`

github: `https://github.com/theoryinpractice/toboggan`

Thank you!