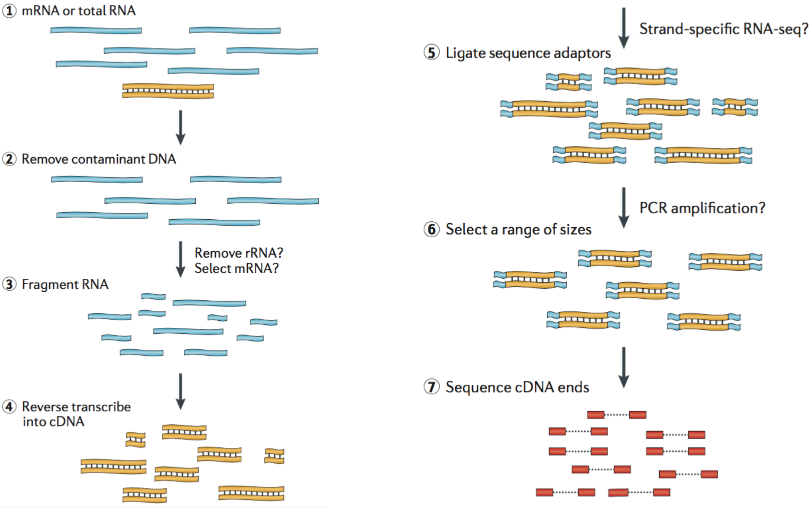
Reference-Based Transcriptome Assembly

Mingfu Shao

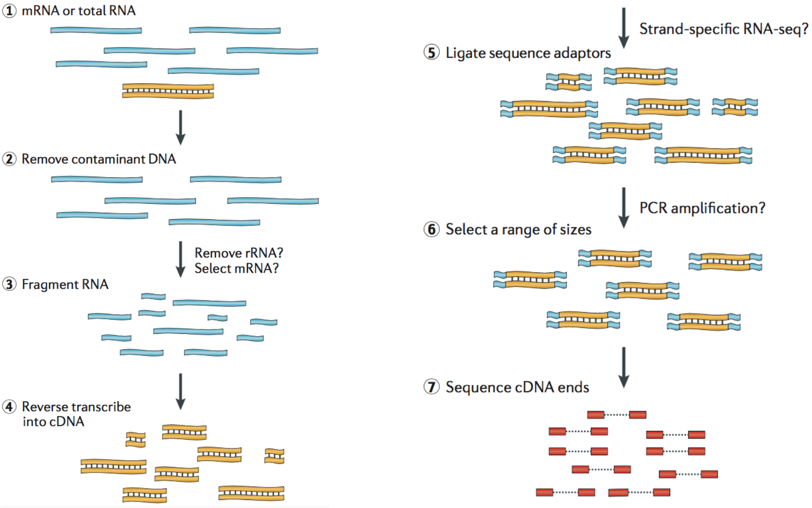
Computational Biology Department, Carnegie Mellon University

January 27, 2016

RNA-seq Experiments



RNA-seq Experiments



Transcriptome Assembly: To determine the transcripts and their abundance from the (paired-end) reads.

Existing Softwares/Methods

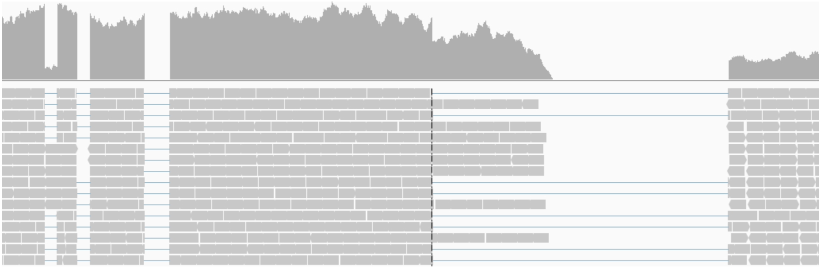
- **Reference-based** methods:
  - Cufflinks (Trapnell *et al.*, 2010)
  - Scripture (Guttman *et al.*, 2010)
  - IsoLasso (Li *et al.*, 2011)
  - SLIDE (Li *et al.*, 2011)
  - CLIIQ (Lin *et al.*, 2012)
  - CEM (Li *et al.*, 2012)
  - MITIE (Behr *et al.*, 2013)
  - Traph (Tomescu *et al.*, 2013)
  - StringTie (Pertea *et al.*, 2015)
  - ...
- ***De novo*** methods:
  - Trans-ABySS (Robertson *et al.*, 2010)
  - Trinity (Grabherr *et al.*, 2011)
  - Oases (Schulz *et al.*, 2012)
  - ...

Existing Softwares/Methods

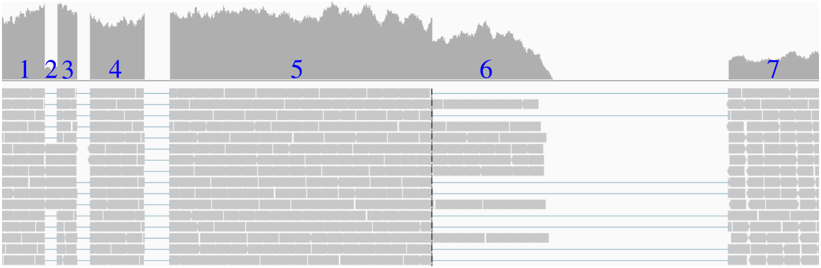
- **Reference-based** methods:
  - Cufflinks (Trapnell *et al.*, 2010) **use overlap graph**
  - Scripture (Guttman *et al.*, 2010)
  - IsoLasso (Li *et al.*, 2011)
  - SLIDE (Li *et al.*, 2011)
  - CLIIQ (Lin *et al.*, 2012)
  - CEM (Li *et al.*, 2012)
  - MITIE (Behr *et al.*, 2013)
  - Traph (Tomescu *et al.*, 2013)
  - StringTie (Pertea *et al.*, 2015)
  - ...

  use splice graph
- ***De novo*** methods:
  - Trans-ABySS (Robertson *et al.*, 2010)
  - Trinity (Grabherr *et al.*, 2011)
  - Oases (Schulz *et al.*, 2012)
  - ...

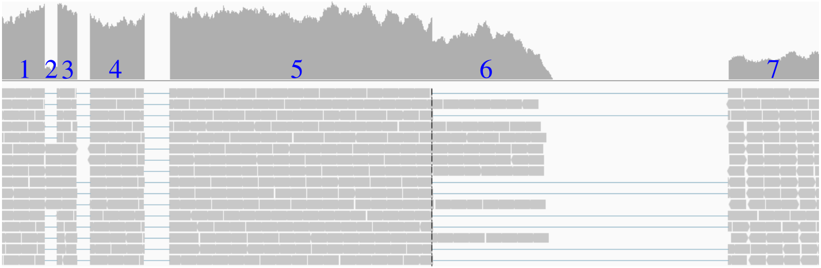
Splice Graph

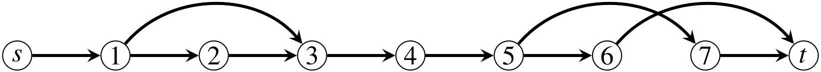


Splice Graph



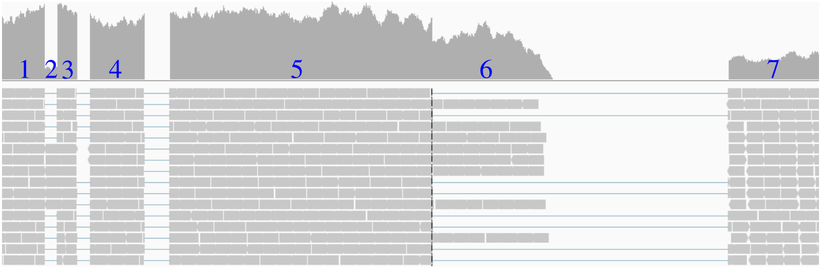
Splice Graph

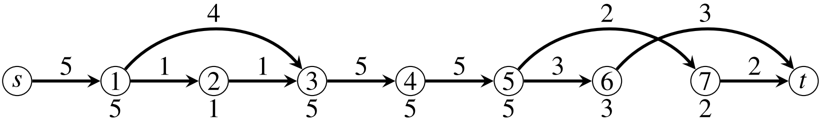




- **Nodes:** continuous regions not seperated by spliced reads
- **Edges:** adjacent regions or connected by spliced reads

Splice Graph

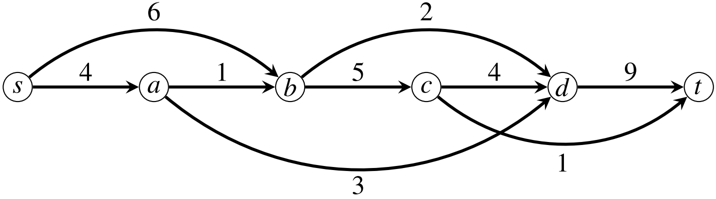




- **Nodes:** continuous regions not seperated by spliced reads
- **Edges:** adjacent regions or connected by spliced reads
- **Weight of nodes:** estimated from the average coverage
- **Weight of edges:** estimated from number of spanning reads

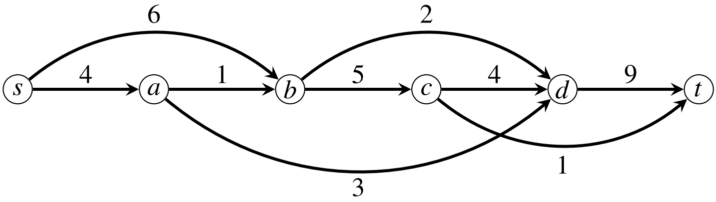
Optimization Problem

- **Input:** Directed acyclic graph (DAG) $G = (V, E)$ with a single source s and a single sink t; weight $w(e)$ for $e \in E$.



Optimization Problem

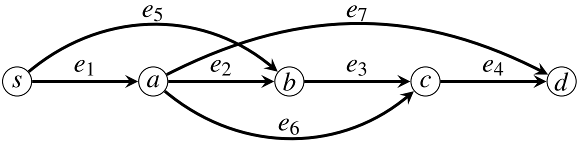
- **Input:** Directed acyclic graph (DAG) $G = (V, E)$ with a single source s and a single sink t; weight $w(e)$ for $e \in E$.



- **Output:** A set of s-t paths $\mathscr{P}$ and abundance $a(P)$ for $P \in \mathscr{P}$, such that
  1. each $e \in E$ is covered by at least one path $P \in \mathscr{P}$, and that
  2. $|\mathscr{P}|$ is as small as possible, and that
  3. $\sum_{e \in E} \|w(e) - \sum_{P: e \in P} a(P)\|$ is as small as possible.

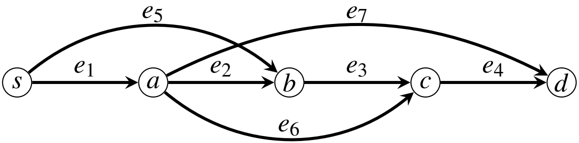
Minimize the Number of Paths

Problem: Given DAG $G = (V, E)$ with source s and sink t, to compute minimum number of paths that can cover all edges.

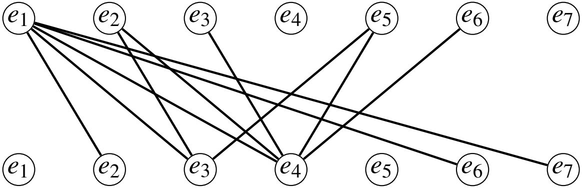


Minimize the Number of Paths

Problem: Given DAG $G = (V, E)$ with source s and sink t, to compute minimum number of paths that can cover all edges.

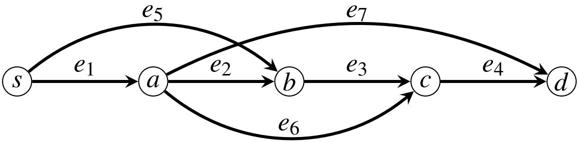


Polynomial-time Algorithm:

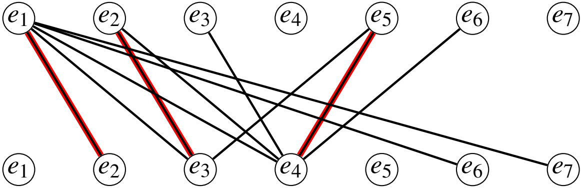


Minimize the Number of Paths

Problem: Given DAG $G = (V, E)$ with source s and sink t, to compute minimum number of paths that can cover all edges.

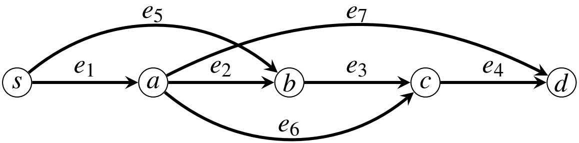


Polynomial-time Algorithm:

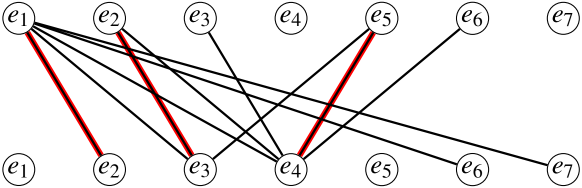


Minimize the Number of Paths

Problem: Given DAG $G = (V, E)$ with source s and sink t, to compute minimum number of paths that can cover all edges.



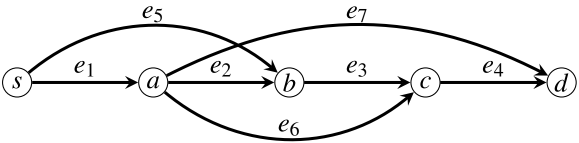
Polynomial-time Algorithm:



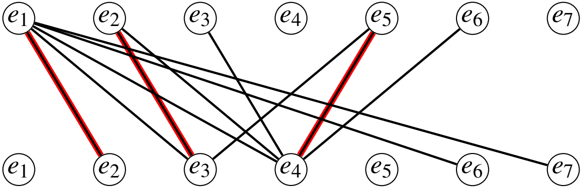
Dilworth's Theorem: $|\mathscr{P}| = |E| - M$.

Minimize the Number of Paths

Problem: Given DAG $G = (V, E)$ with source s and sink t, to compute minimum number of paths that can cover all edges.



Polynomial-time Algorithm:



Dilworth's Theorem: $|\mathscr{P}| = |E| - M$.

Existing Software: Cufflinks (Trapnell *et al.*, 2010)

Minimize Abundance Error

Problem: Given DAG $G = (V, E)$ with source s, sink t, and weight $w(e)$ for $e \in E$, to compute a set of s-t paths $\mathscr{P}$ so that all edges are covered and that $\sum_{e \in E} |w(e) - \sum_{P: e \in P} a(P)|$ is minimized.

Minimize Abundance Error

Problem: Given DAG $G = (V, E)$ with source s, sink t, and weight $w(e)$ for $e \in E$, to compute a set of s-t paths $\mathscr{P}$ so that all edges are covered and that $\sum_{e \in E} |w(e) - \sum_{P : e \in P} a(P)|$ is minimized.

Fact: There exists a set of paths satisfying $x(e) = \sum_{P \in \mathscr{P} : e \in P} a(P)$ for all $e \in E$ if and only if $x(e)$ forms a flow of G.

Minimize Abundance Error

Problem: Given DAG $G = (V, E)$ with source s, sink t, and weight $w(e)$ for $e \in E$, to compute a set of s-t paths $\mathscr{P}$ so that all edges are covered and that $\sum_{e \in E} |w(e) - \sum_{P: e \in P} a(P)|$ is minimized.

Fact: There exists a set of paths satisfying $x(e) = \sum_{P \in \mathscr{P}: e \in P} a(P)$ for all $e \in E$ if and only if $x(e)$ forms a flow of G.

Reformulation: To compute $x(e)$ for $e \in E$ such that $x(e)$ forms a flow and that $\sum_{e \in E} |w(e) - x(e)|$ is minimized.

Minimize Abundance Error

Problem: Given DAG $G=(V,E)$ with source s, sink t, and weight $w(e)$ for $e \in E$, to compute a set of s-t paths $\mathscr{P}$ so that all edges are covered and that $\sum_{e\in E} |w(e) - \sum_{P:e\in P} a(P)|$ is minimized.

Fact: There exists a set of paths satisfying $x(e) = \sum_{P\in\mathscr{P}:e\in P} a(P)$ for all $e \in E$ if and only if $x(e)$ forms a flow of G.

Reformulation: To compute $x(e)$ for $e \in E$ such that $x(e)$ forms a flow and that $\sum_{e\in E} |w(e) - x(e)|$ is minimized.

Polynomial-time Algorithm (using LP):

$$\begin{array}{ll} \min & \sum_{e\in E} |w(e) - x(e)| \\ \text{s.t.} & \begin{cases} \sum_{e=(u,v)\in E} x(e) = \sum_{e=(v,w)\in E} x(e), \forall v \in V \\ x(e) \geq 0, \forall e \in E \end{cases} \end{array}$$

Minimize Abundance Error

Problem: Given DAG $G = (V, E)$ with source s, sink t, and weight $w(e)$ for $e \in E$, to compute a set of s-t paths $\mathscr{P}$ so that all edges are covered and that $\sum_{e \in E} |w(e) - \sum_{P: e \in P} a(P)|$ is minimized.

Fact: There exists a set of paths satisfying $x(e) = \sum_{P \in \mathscr{P}: e \in P} a(P)$ for all $e \in E$ if and only if $x(e)$ forms a flow of G.

Reformulation: To compute $x(e)$ for $e \in E$ such that $x(e)$ forms a flow and that $\sum_{e \in E} |w(e) - x(e)|$ is minimized.

Polynomial-time Algorithm (using LP):

$$\begin{array}{ll} \min & \sum_{e \in E} |w(e) - x(e)| \\ \text{s.t.} & \left\{ \begin{array}{l} \sum_{e=(u,v) \in E} x(e) = \sum_{e=(v,w) \in E} x(e), \forall v \in V \\ x(e) \geq 0, \forall e \in E \end{array} \right. \end{array}$$

Based on $x(e)$, **greedy algorithm** can be used to retrieve $\mathscr{P}$ with at most $|E| - |V| + 2$ paths.

Minimize Abundance Error

Problem: Given DAG $G = (V, E)$ with source s, sink t, and weight $w(e)$ for $e \in E$, to compute a set of s-t paths $\mathscr{P}$ so that all edges are covered and that $\sum_{e \in E} |w(e) - \sum_{P: e \in P} a(P)|$ is minimized.

Fact: There exists a set of paths satisfying $x(e) = \sum_{P \in \mathscr{P}: e \in P} a(P)$ for all $e \in E$ if and only if $x(e)$ forms a flow of G.

Reformulation: To compute $x(e)$ for $e \in E$ such that $x(e)$ forms a flow and that $\sum_{e \in E} |w(e) - x(e)|$ is minimized.

Polynomial-time Algorithm (using LP):

$$\begin{array}{ll} \min & \sum_{e \in E} |w(e) - x(e)| \\ \text{s.t.} & \begin{cases} \sum_{e=(u,v) \in E} x(e) = \sum_{e=(v,w) \in E} x(e), \forall v \in V \\ x(e) \geq 0, \forall e \in E \end{cases} \end{array}$$

Based on $x(e)$, **greedy algorithm** can be used to retrieve $\mathscr{P}$ with at most $|E| - |V| + 2$ paths.

Existing Software: Traph (Tomescu *et al.*, 2013)

Minimize Abundance Error with k Paths

Problem: Given DAG $G=(V,E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ such that $\sum_{e\in E} \|w(e) - \sum_{P:e\in P} a(P)\|$ is minimized.

Minimize Abundance Error with k Paths

Problem: Given DAG $G = (V, E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ such that $\sum_{e \in E} \|w(e) - \sum_{P: e \in P} a(P)\|$ is minimized.

Maximum k-Splittable Flow Problem: Given DAG $G = (V, E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ satisfying $\sum_{P: e \in P} a(P) \le w(e), \forall e \in E$ (capacity constraints) such that $\sum_{P \in \mathscr{P}} a(P)$ is maximized.

Minimize Abundance Error with k Paths

Problem: Given DAG $G=(V,E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ such that $\sum_{e\in E} \|w(e) - \sum_{P:e\in P} a(P)\|$ is minimized.

Maximum k-Splittable Flow Problem: Given DAG $G=(V,E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ satisfying $\sum_{P:e\in P} a(P) \leq w(e), \forall e \in E$ (capacity constraints) such that $\sum_{P\in\mathscr{P}} a(P)$ is maximized.

1. MkSF is **NP**-hard for $2 \leq k < |E| - |V| + 2$.

Minimize Abundance Error with k Paths

Problem: Given DAG $G=(V,E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ such that $\sum_{e\in E} \|w(e) - \sum_{P:e\in P} a(P)\|$ is minimized.

Maximum k-Splittable Flow Problem: Given DAG $G=(V,E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ satisfying $\sum_{P:e\in P} a(P) \le w(e), \forall e \in E$ (capacity constraints) such that $\sum_{P\in\mathscr{P}} a(P)$ is maximized.

1. MkSF is **NP**-hard for $2 \le k < |E| - |V| + 2$.
2. 1.5-approximation algorithm for $k=2$ and $k=3$.

Minimize Abundance Error with k Paths

Problem: Given DAG $G = (V, E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ such that $\sum_{e \in E} \|w(e) - \sum_{P: e \in P} a(P)\|$ is minimized.

Maximum k-Splittable Flow Problem: Given DAG $G = (V, E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ satisfying $\sum_{P: e \in P} a(P) \leq w(e), \forall e \in E$ (capacity constraints) such that $\sum_{P \in \mathscr{P}} a(P)$ is maximized.

1. MkSF is **NP**-hard for $2 \leq k < |E| - |V| + 2$.
2. 1.5-approximation algorithm for $k = 2$ and $k = 3$.
3. 2-approximation algorithm for $4 \leq k < |E| - |V| + 2$.

Minimize Abundance Error with k Paths

Problem: Given DAG $G=(V,E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ such that $\sum_{e\in E} \|w(e) - \sum_{P:e\in P} a(P)\|$ is minimized.

Maximum k-Splittable Flow Problem: Given DAG $G=(V,E)$ with source s, sink t, weight $w(e)$ for $e \in E$, to compute a set of at most k s-t paths $\mathscr{P}$ satisfying $\sum_{P:e\in P} a(P) \le w(e), \forall e \in E$ (capacity constraints) such that $\sum_{P\in\mathscr{P}} a(P)$ is maximized.

1. MkSF is **NP**-hard for $2 \le k < |E| - |V| + 2$.
2. 1.5-approximation algorithm for $k=2$ and $k=3$.
3. 2-approximation algorithm for $4 \le k < |E| - |V| + 2$.
4. Polynomial-time algorithm for G with constant treewidth.

Regularization Methods

Optimization Formulation:

$$\min \sum_{e \in E} \|w(e) - \sum_{P: e \in P} a(P)\| + \lambda \sum_{P \in \mathscr{P}} \|a(P)\|^q$$

Regularization Methods

Optimization Formulation:

$$\min \sum_{e \in E} \|w(e) - \sum_{P: e \in P} a(P)\| + \lambda \sum_{P \in \mathscr{P}} \|a(P)\|^q$$

1. Initialize $\mathscr{P}$ as all possible paths (exponential size).
2. Parameter λ needs to be trained.
3. Use continuous optimization techniques (Lasso, Newton–Raphson, etc) to compute a local optimal solution.

Regularization Methods

Optimization Formulation:

$$\min \sum_{e \in E} \|w(e) - \sum_{P: e \in P} a(P)\| + \lambda \sum_{P \in \mathscr{P}} \|a(P)\|^q$$

1. Initialize $\mathscr{P}$ as all possible paths (exponential size).
2. Parameter λ needs to be trained.
3. Use continuous optimization techniques (Lasso, Newton–Raphson, etc) to compute a local optimal solution.

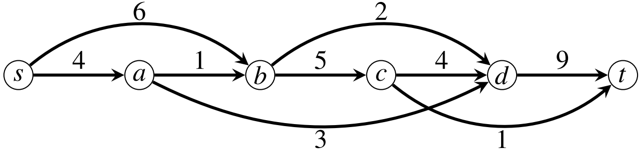
Existing Softwares: IsoLasso (Li *et al.*, 2011), SLIDE (Li *et al.*, 2011), CEM (Li *et al.*, 2012), MITIE (Behr *et al.*, 2013)

Heuristic—Greedy Algorithm

Algorithm: Iteratively compute the path with maximum bottleneck weight, and then update the graph.

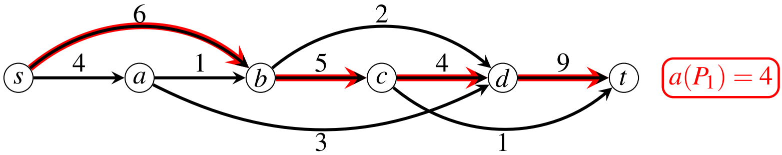
Heuristic—Greedy Algorithm

Algorithm: Iteratively compute the path with maximum bottleneck weight, and then update the graph.



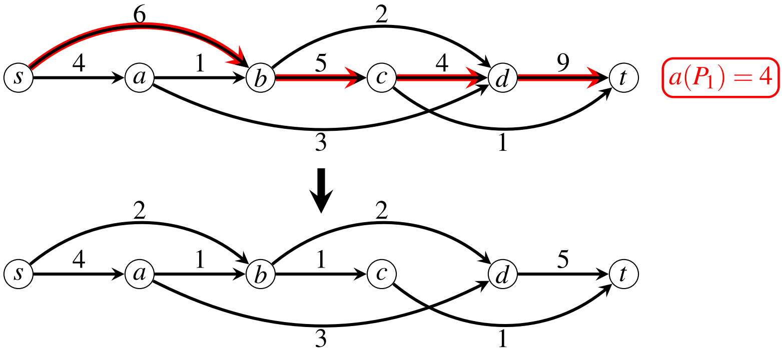
Heuristic—Greedy Algorithm

Algorithm: Iteratively compute the path with maximum bottleneck weight, and then update the graph.



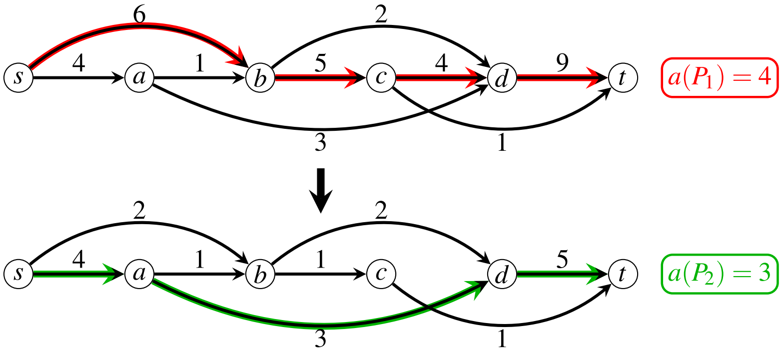
Heuristic—Greedy Algorithm

Algorithm: Iteratively compute the path with maximum bottleneck weight, and then update the graph.



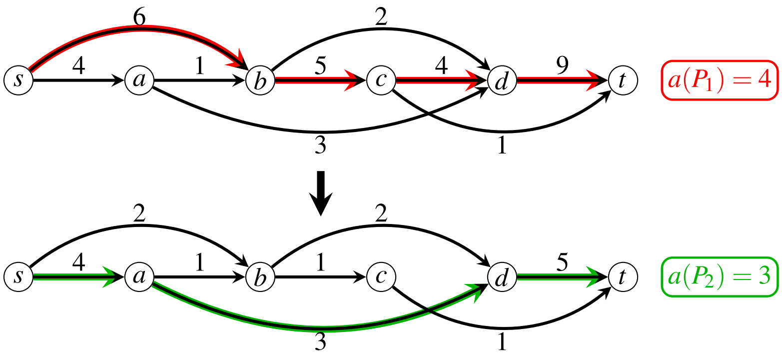
Heuristic—Greedy Algorithm

Algorithm: Iteratively compute the path with maximum bottleneck weight, and then update the graph.



Heuristic—Greedy Algorithm

Algorithm: Iteratively compute the path with maximum bottleneck weight, and then update the graph.



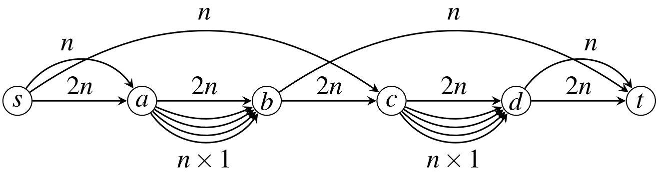
Existing Softwares: Traph (Tomescu *et al.*, 2013), StringTie (Pertea *et al.*, 2015)

Our Method—Framework

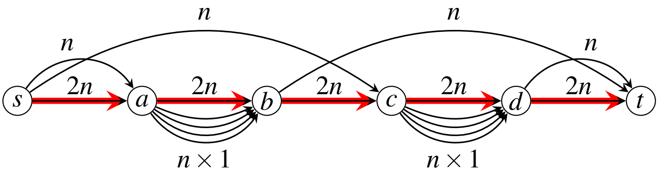
1. Initialize a set of paths $\mathscr{P}$ (in polynomial size).
2. Use LP to compute $a(P)$ for $P \in \mathscr{P}$ so as to minimize the estimation error.
3. Reduce the number of paths in $\mathscr{P}$:
   - For two paths with (almost) identical abundance, try to merge them into one path;
   - Discard paths with very small abundance.
4. Iterate between step 2 and step 3.

Example for Greedy Algorithm

Example for Greedy Algorithm

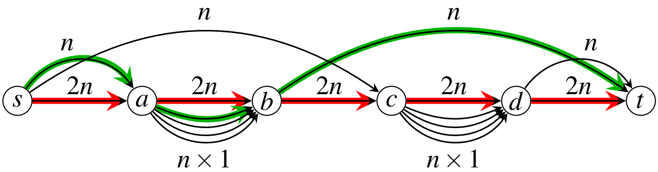


Example for Greedy Algorithm



$\boxed{a(P_1) = 2n\ (\times 1)}$

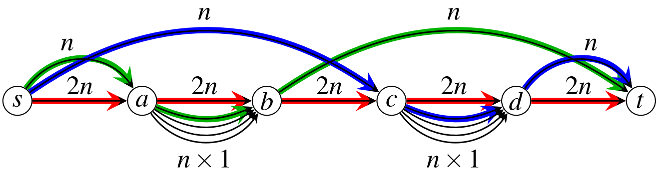
Example for Greedy Algorithm



$\boxed{a(P_1) = 2n\ (\times 1)}$

$\boxed{a(P_2) = 1\ (\times n)}$

Example for Greedy Algorithm

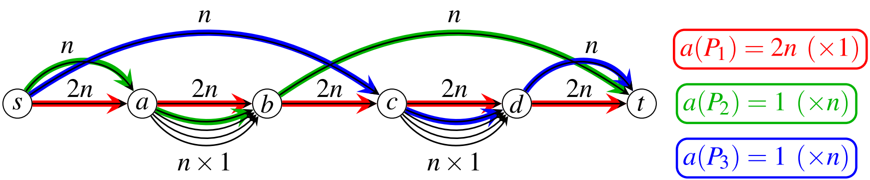


$a(P_1) = 2n\ (\times 1)$

$a(P_2) = 1\ (\times n)$

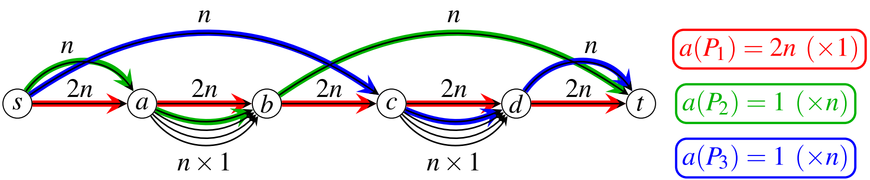
$a(P_3) = 1\ (\times n)$

Example for Greedy Algorithm

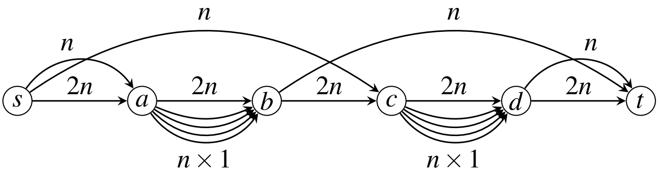


Solution by greedy algorithm: $2n + 1$ paths.

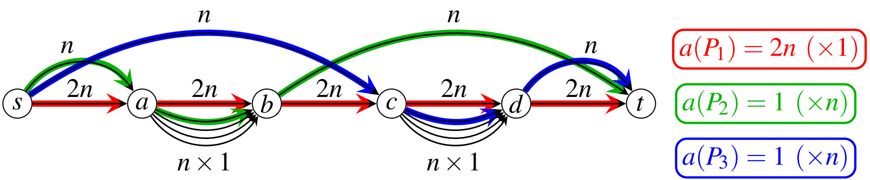
Example for Greedy Algorithm



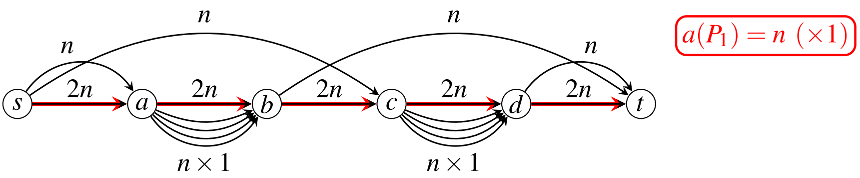
Solution by greedy algorithm: $2n+1$ paths.



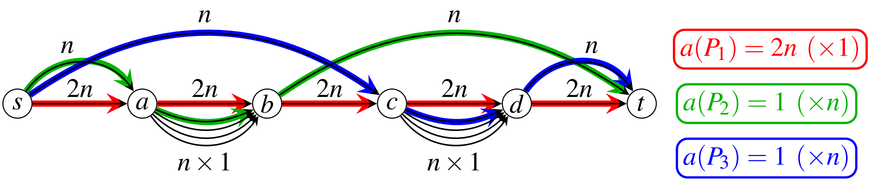
Example for Greedy Algorithm



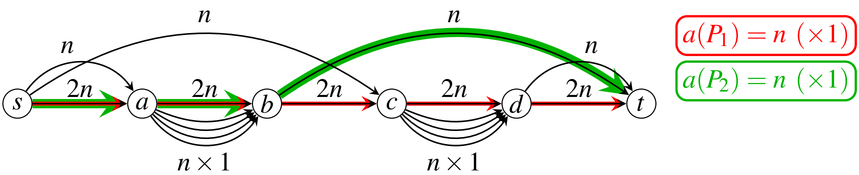
Solution by greedy algorithm: $2n+1$ paths.



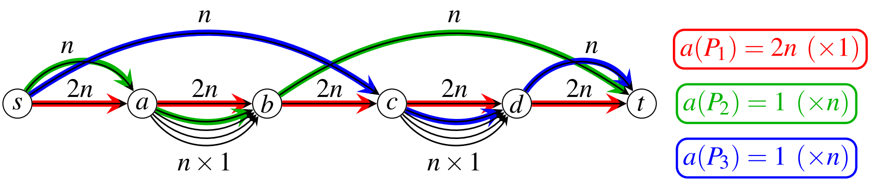
Example for Greedy Algorithm



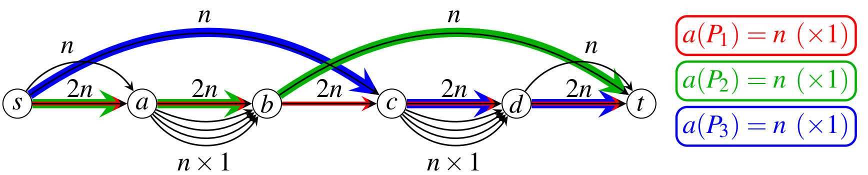
Solution by greedy algorithm: $2n + 1$ paths.



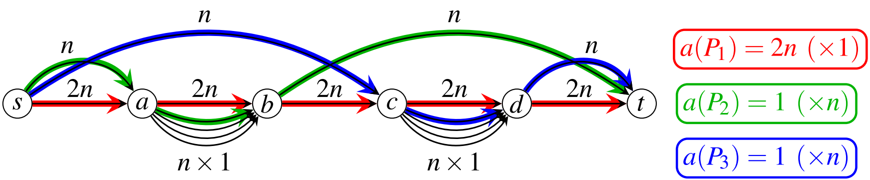
Example for Greedy Algorithm



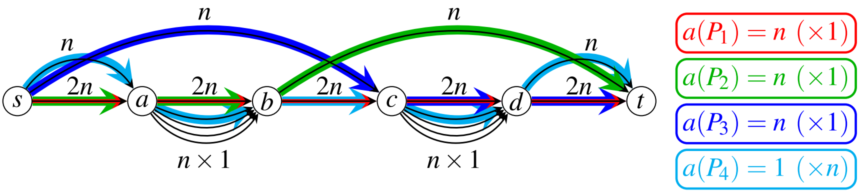
Solution by greedy algorithm: $2n+1$ paths.



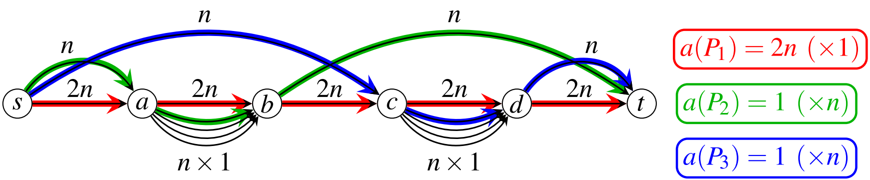
Example for Greedy Algorithm



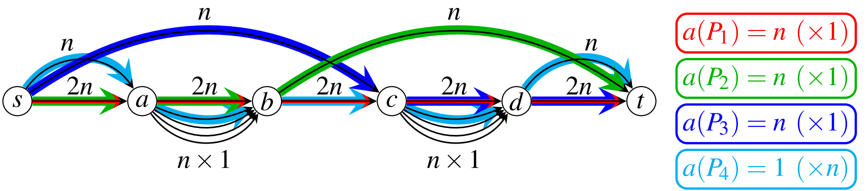
Solution by greedy algorithm: $2n+1$ paths.



Example for Greedy Algorithm



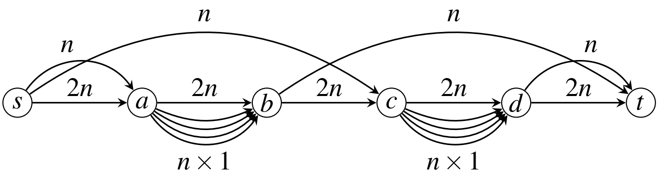
Solution by greedy algorithm: $2n + 1$ paths.



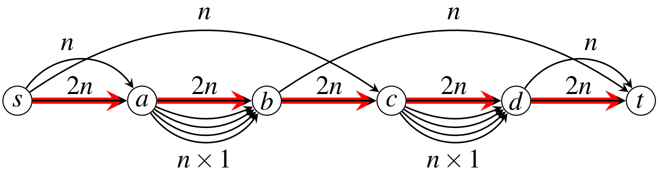
Optimal solution: $n + 3$ paths.

Use the Residual Graph

Use the Residual Graph

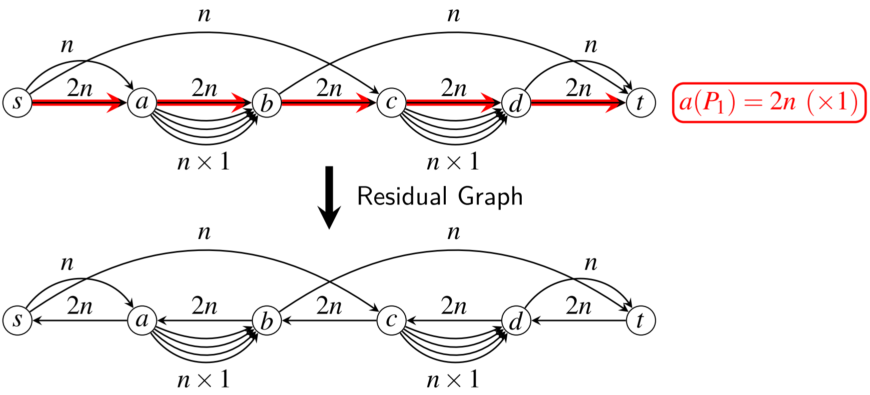


Use the Residual Graph

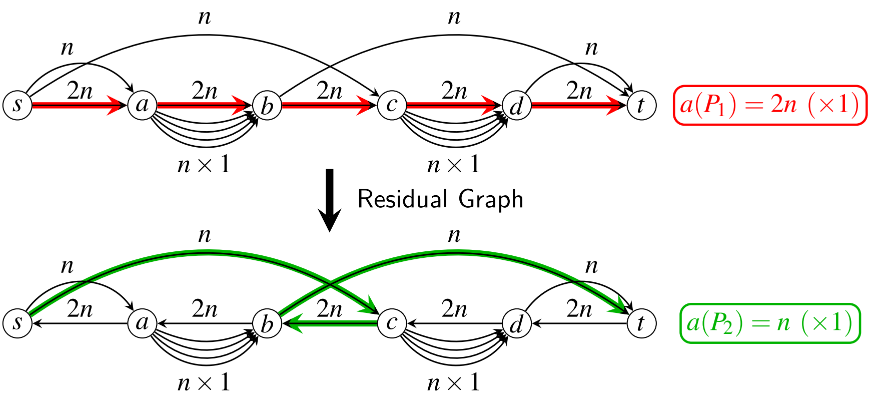


$\boxed{a(P_1) = 2n \ (\times 1)}$

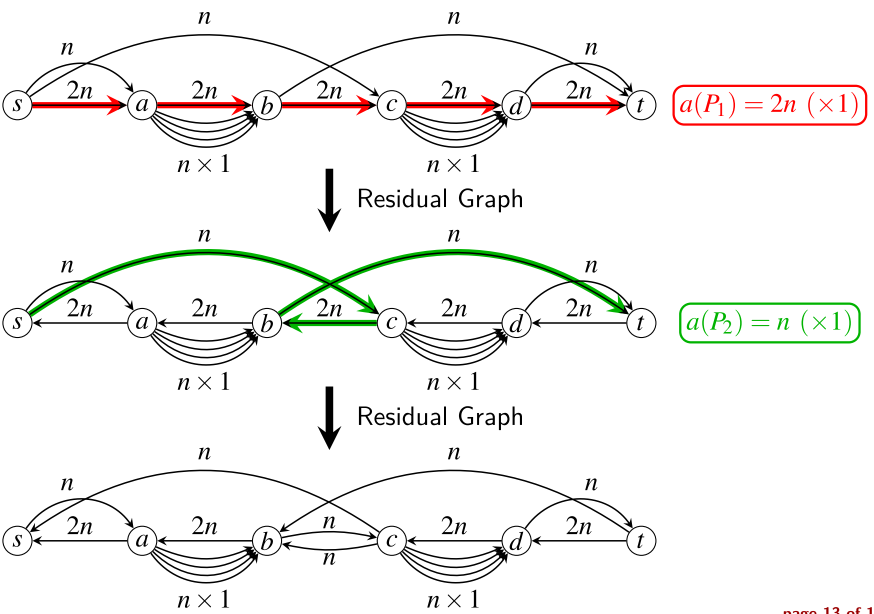
Use the Residual Graph



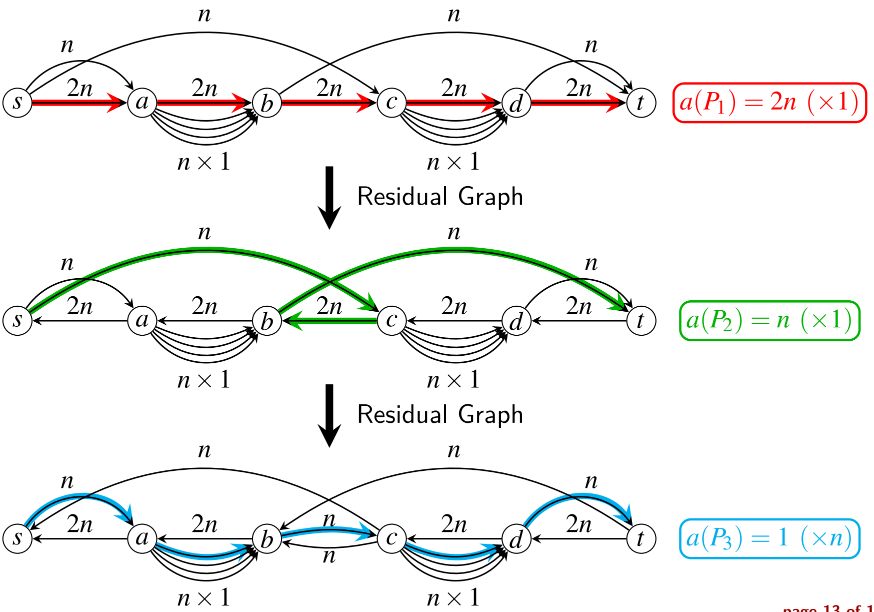
Use the Residual Graph



Use the Residual Graph



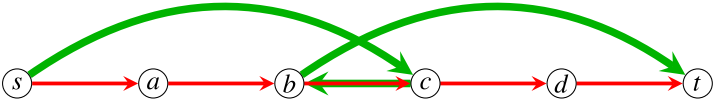
Use the Residual Graph



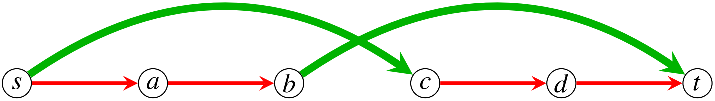
Resolve Path with Backward Edges

Resolve Path with Backward Edges

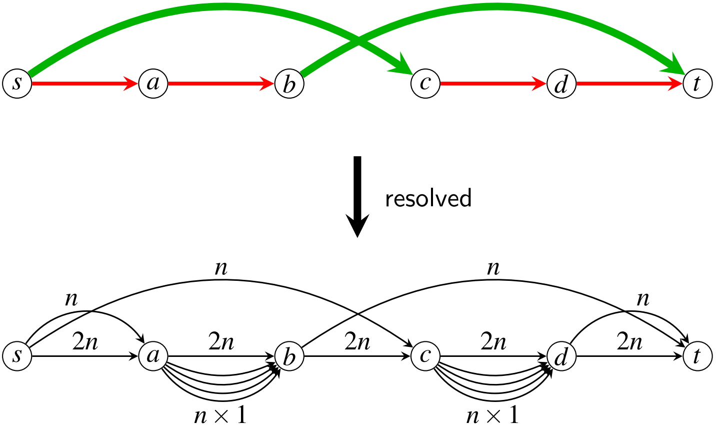
Resolve Path with Backward Edges



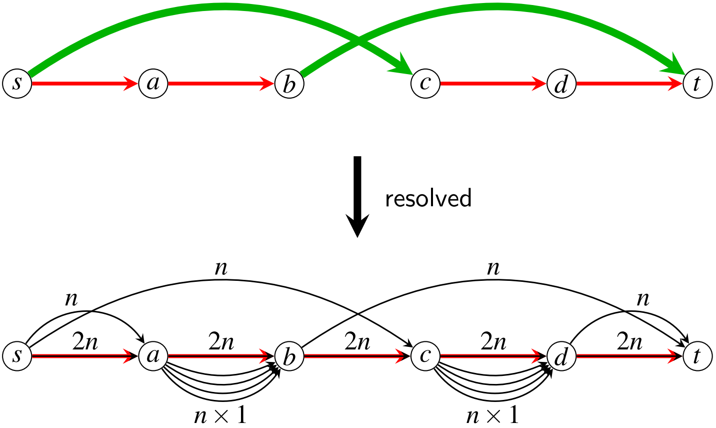
Resolve Path with Backward Edges



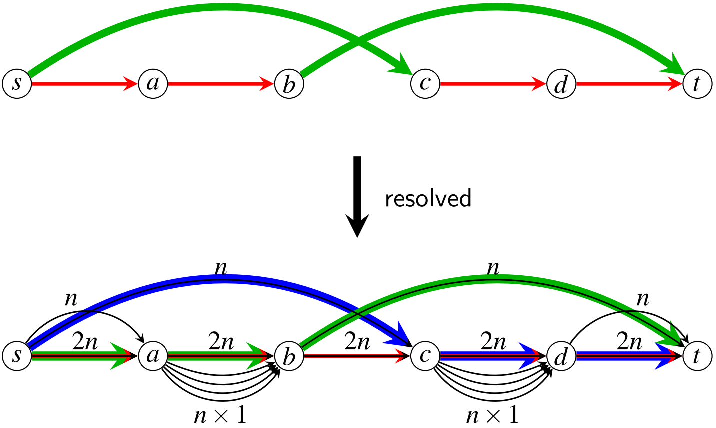
Resolve Path with Backward Edges



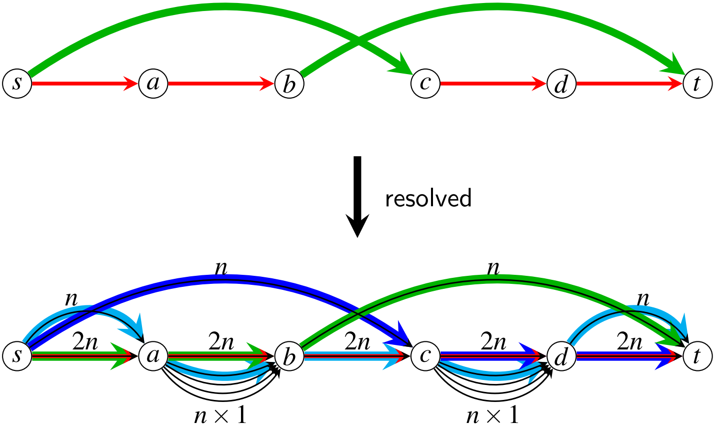
Resolve Path with Backward Edges



Resolve Path with Backward Edges



Resolve Path with Backward Edges



Algorithm to Initialize $\mathscr{P}$

1. Initialize $\mathscr{P} = \emptyset$.
2. Compute a set of paths $\mathscr{P}'$ (paths in $\mathscr{P}'$ may contain backward edges) following the max-flow algorithm.
3. **For** each $P \in \mathscr{P}'$ in its original order:
    1. **If** P does not contain backward edges, let $\mathscr{P} = \mathscr{P} \cup \{P\}$;
    2. **else** resolve P using $\mathscr{P}$ and put all resulting paths into $\mathscr{P}$.
    3. **If** $\mathscr{P}$ becomes a basis, or $|\mathscr{P}| \geq C$, **break**.
4. **Return** $\mathscr{P}$.

Our Method—Framework

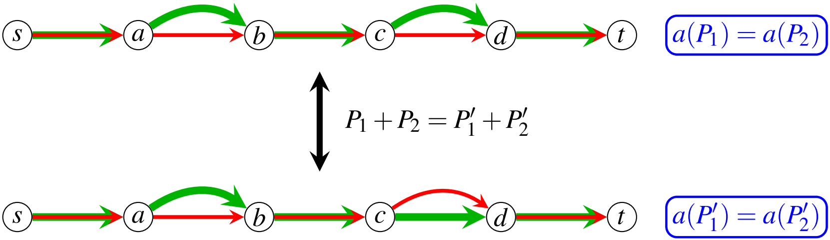
1. Initialize a set of paths $\mathscr{P}$ (in polynomial size).
2. Use LP to compute $a(P)$ for $P \in \mathscr{P}$ so as to minimize the estimation error.
3. Reduce the number of paths in $\mathscr{P}$:
   - For two paths with (almost) identical abundance, try to merge them into one path;
   - Discard paths with very small abundance.
4. Iterate between step 2 and step 3.

Merge Paths: Example

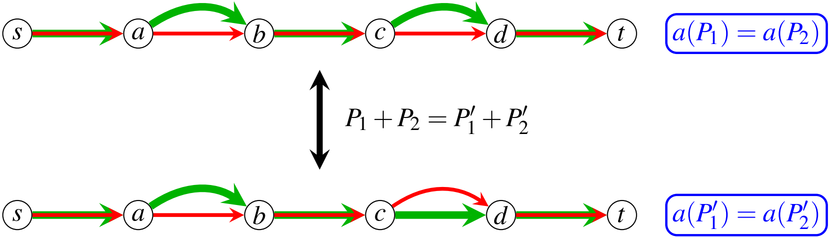
Merge Paths: Example

$\boxed{a(P_1) = a(P_2)}$

Merge Paths: Example



Merge Paths: Example



$$\mathscr{P} = \begin{cases} P_1: & a(P_1) = p \\ P_2: & a(P_2) = p \\ P_1': & a(P_1') = q \\ \dots & \end{cases} \Longrightarrow \mathscr{P}' = \begin{cases} P_1': & a(P_1') = q + p \\ P_2': & a(P_2') = p \\ \dots & \end{cases}$$