AMSC 663 Mid-Year Presentation

OAR Lib: An Open Source, Arc Routing Library

by Oliver Lum,

Advised by
Professor B. Golden

Overview

- We seek to tackle the fundamental Arc-Routing Problems:
  - The Chinese Postman Problem: Given a graph G, (may be undirected, directed, mixed, or windy), find a least cost traversal of all the links in the graph that returns to the starting node (closed).
  - The Rural Postman Problem: Given a graph G (directed for our purposes), find a least cost traversal of all the *required* links in the graph that returns to the starting node (closed).

Overview

- The general strategy will be to attempt to augment the graph in a least cost way to reach an Eulerian graph, over which an Euler circuit may be constructed efficiently.

Graph	HashMap<Integer, Vertex> internalVertexMap	HashMap<Integer,Vertex> globalVertexMap
	HashMap<Integer, Edge> internalEdgeMap	HashMap<Integer, Edge> globalEdgeMap
Vertex	int mID	int matchID
	int globalID	int mDemand
	boolean demandSet	
Link	int mID	int matchId
	int globalID	int mCapacity
	int mCost	boolean isDirected
	boolean capacitySet	Pair<Vertex> endpoints

Graph

- Directed
- Undirected
- Mixed
- Windy

Vertex

- Directed
  - In/Out Degree
  - Asymmetric Neighbors
- Undirected
  - Degree
  - Symmetric Neighbors
- Mixed
  - Supports Both

Link

- Arc
  - getHead();
  - getTail();
  - isDirected() = true;
- Edge
  - isDirected() = false;
- Mixed Edge
  - Supports Both
- Windy Edge
  - Asymmetric costs.

Solver

- DCPP
  - Exact
- UCPP
  - Exact
- MCPP
  - Frederickson's
  - Yaoyuenyong's
- WPP
  - Win's
  - Benavent's

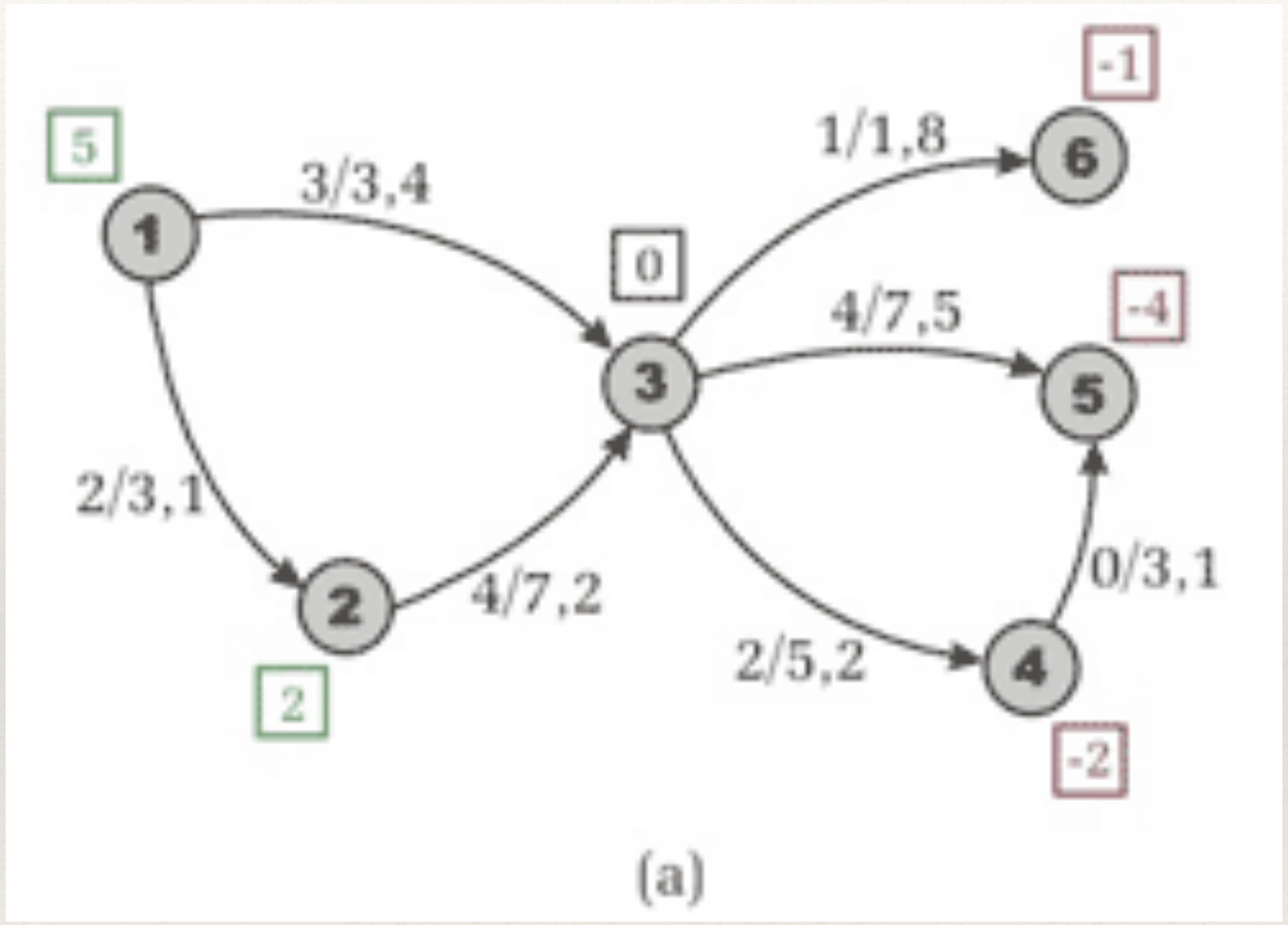
Common Algorithms

- There are several algorithms which are prevalent in almost all of our heuristic solvers:
  - Min-Cost Flow:
    - Cycle-canceling min-cost flow
    - Shortest successive paths min-cost flow.
  - Single-Source Shortest Paths
    - Dijkstra's
    - Bellman-Ford (Not implemented yet)
  - All-Pairs Shortest Paths
    - Floyd-Warshall

Cycle Canceling Min-Cost Flow

- Establish initial feasible flow (greedily).
- Set up the residual graph.
- Detect and delete negative cycles by pushing flow around them.
- Halt when no negative cycles exist.

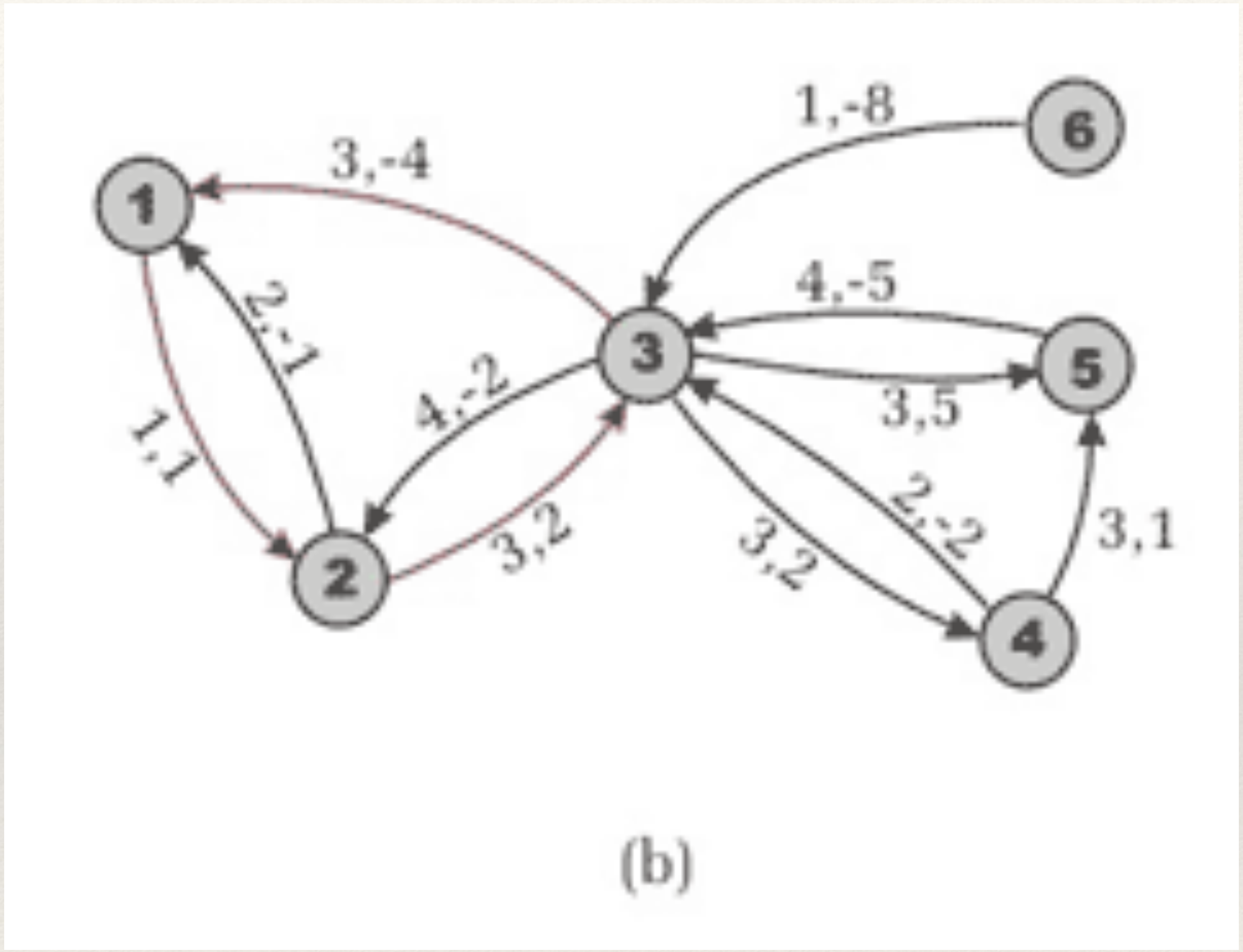
Cycle Canceling Min-Cost Flow



Example of flow problem with initial feasible (non-optimal) solution

http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=minimumCostFlow2

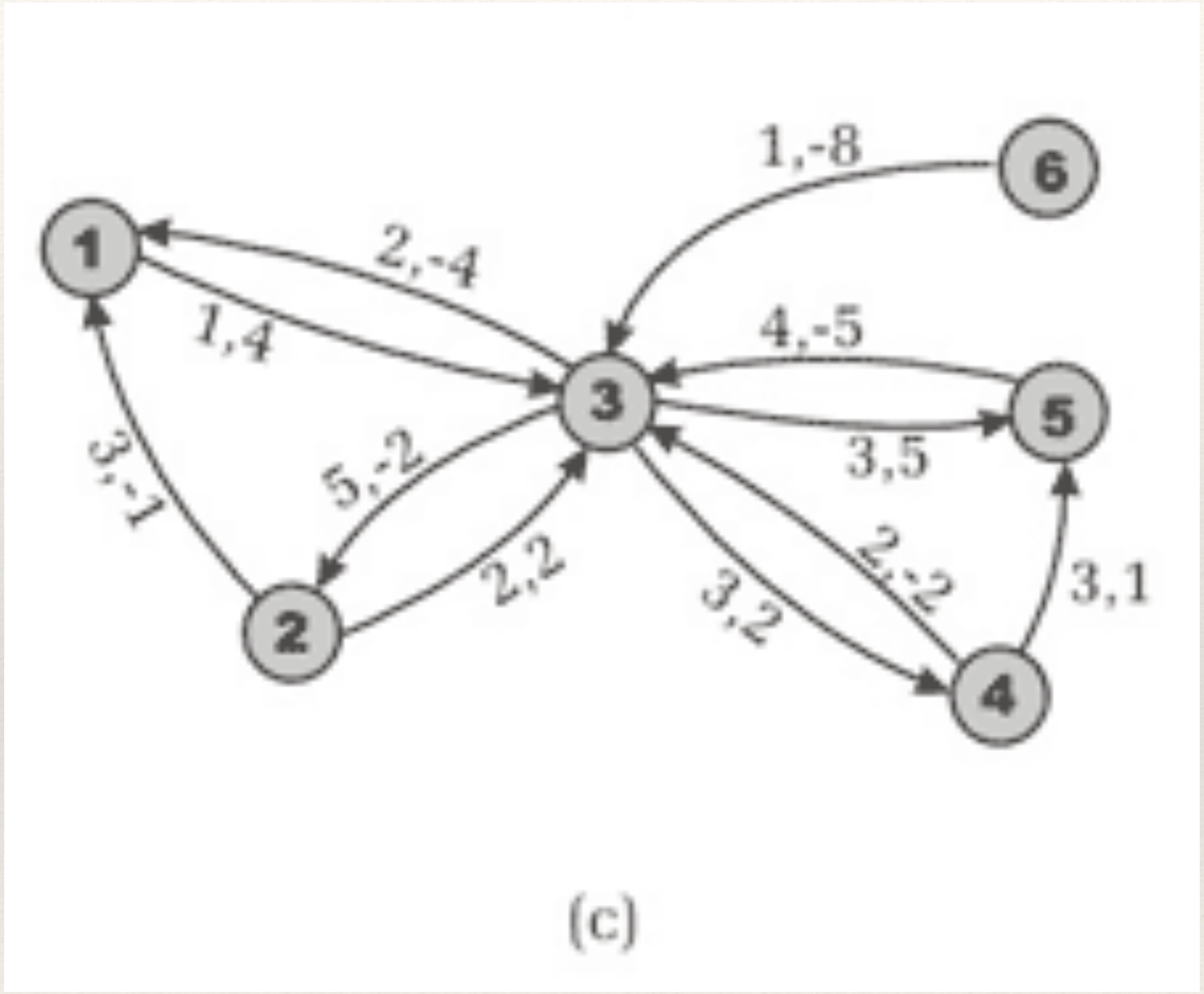
Cycle Canceling Min-Cost Flow



Negative cycle detected in the residual graph; 1 unit of flow pushed around to cancel it.

http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=minimumCostFlow2

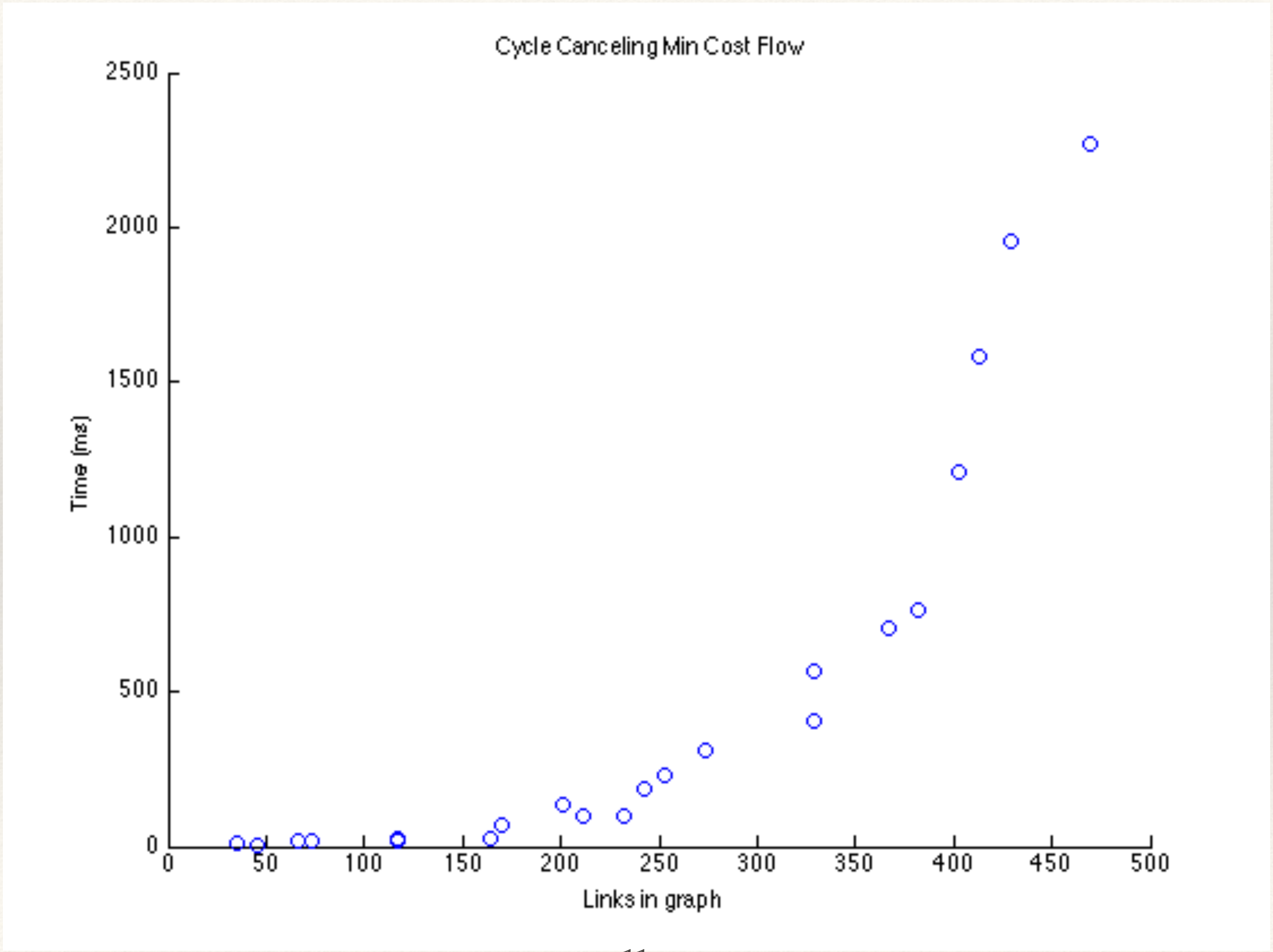
Cycle Canceling Min-Cost Flow



The network after the negative cycle has been cancelled.

http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=minimumCostFlow2

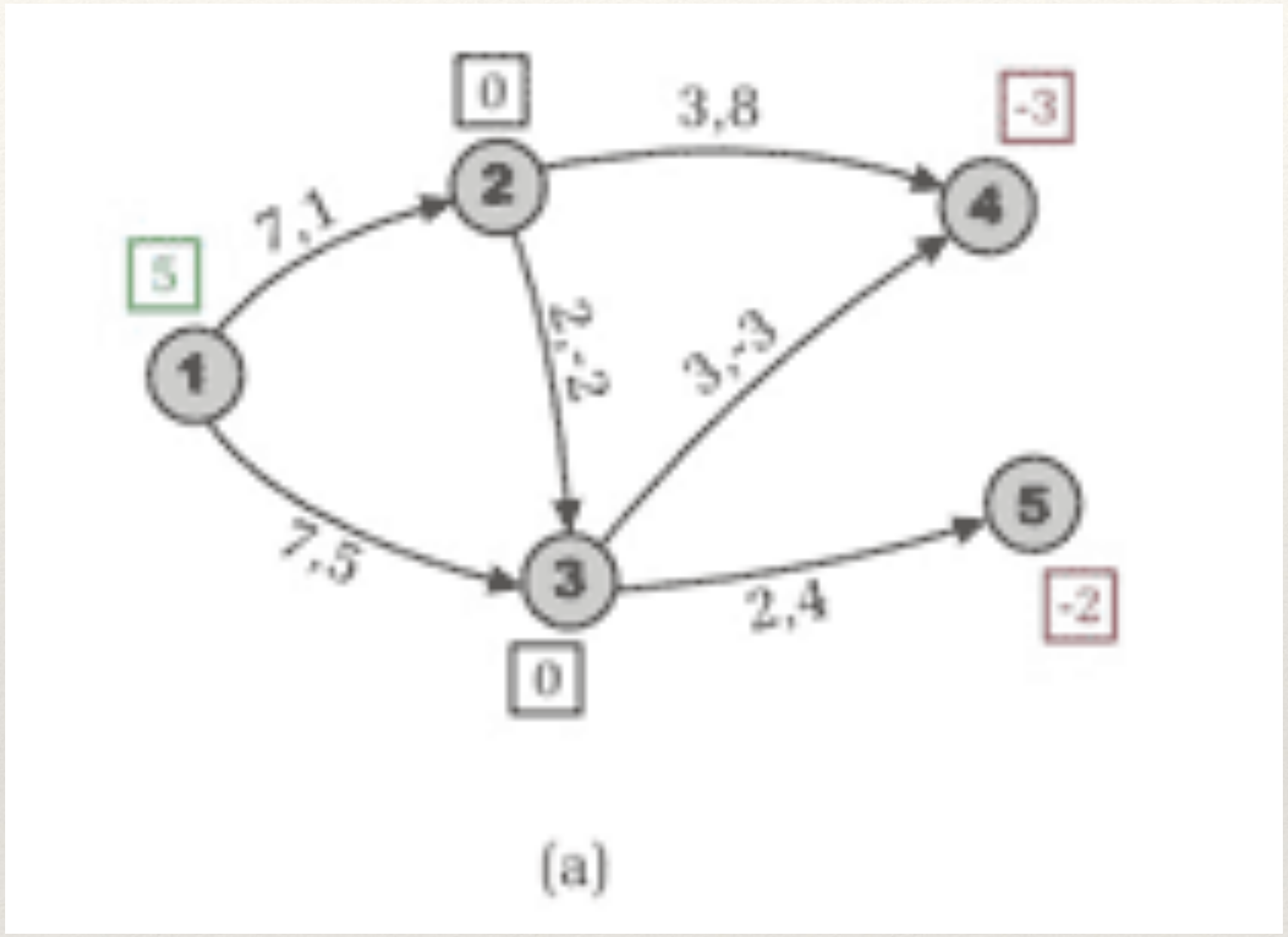
Cycle Canceling Min-Cost Flow



Successive Shortest Paths Min-Cost Flow

- Add an artificial “Source” and “Sink” node to the graph.
- Calculate a shortest path from Source to Sink, and push as much flow as possible from Source to Sink.
- Form the residual graph with the new flow.
- Repeat until no paths from Source to Sink exist.

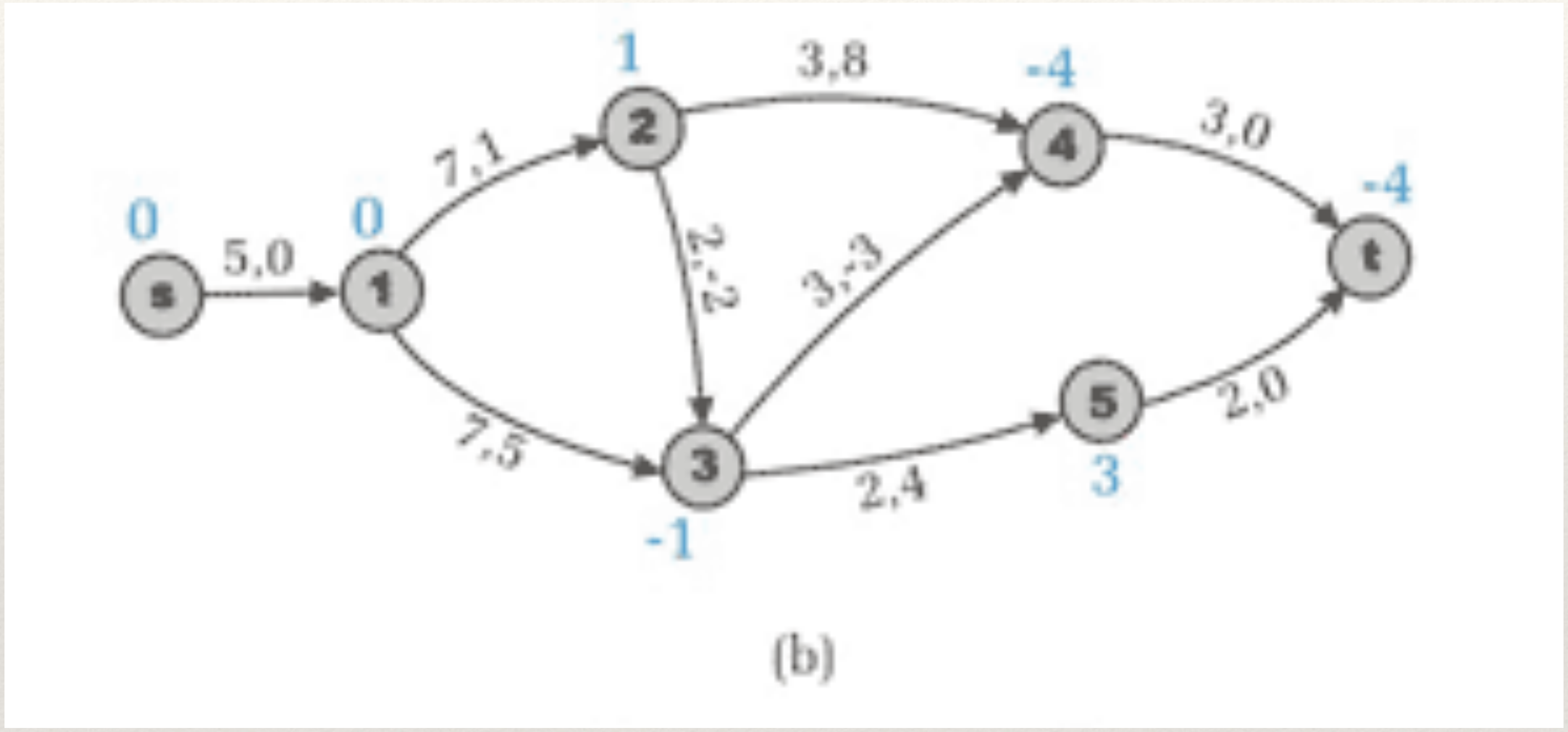
Successive Shortest Paths Min-Cost Flow



The original flow problem.

http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=minimumCostFlow2

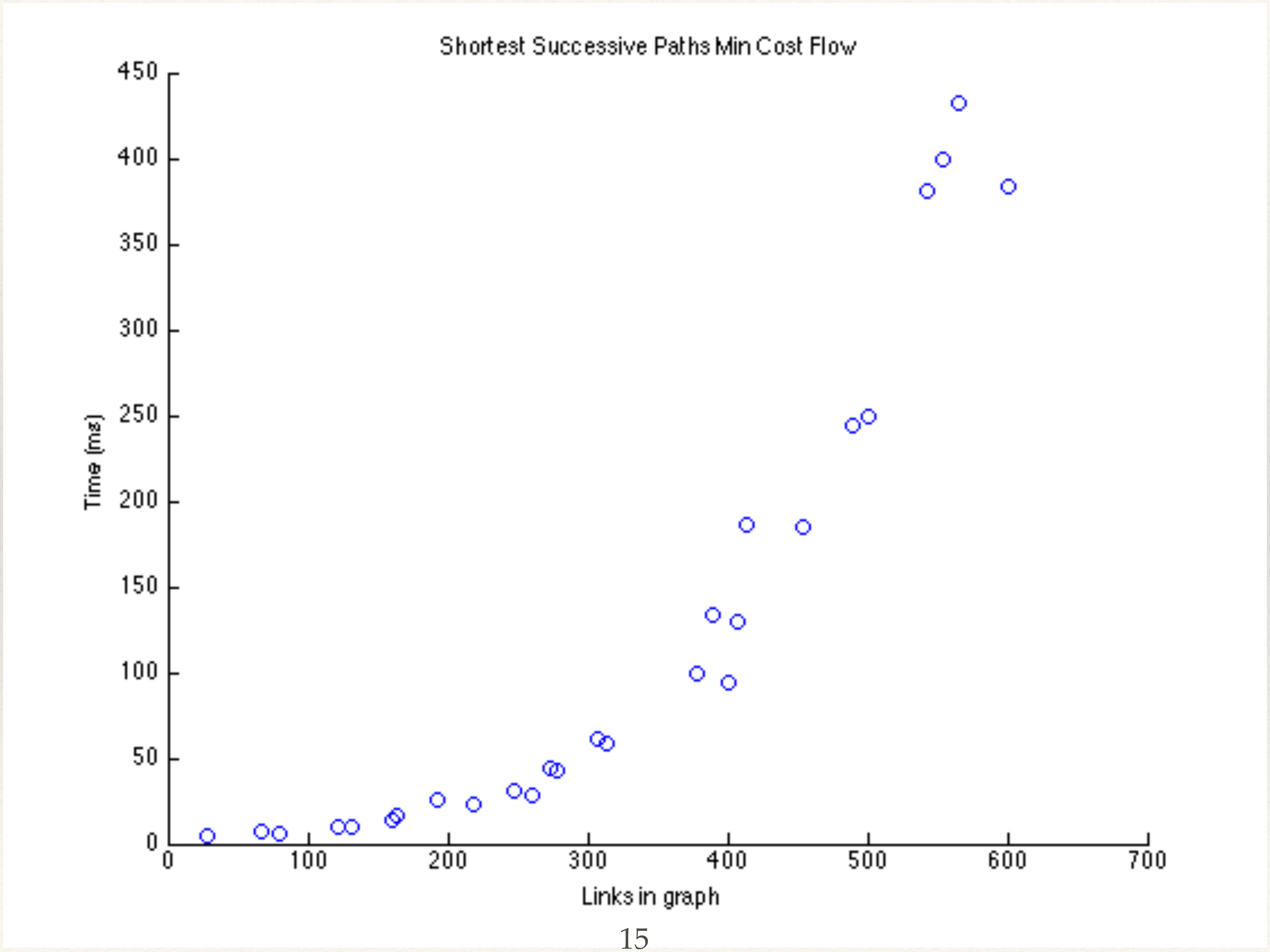
Successive Shortest Paths Min-Cost Flow



The modified network on which the SSP algorithm operates. (Node potentials in Blue).

http://community.topcoder.com/tc?module=Static&d1=tutorials&d2=minimumCostFlow2

Successive Shortest Paths Min-Cost Flow



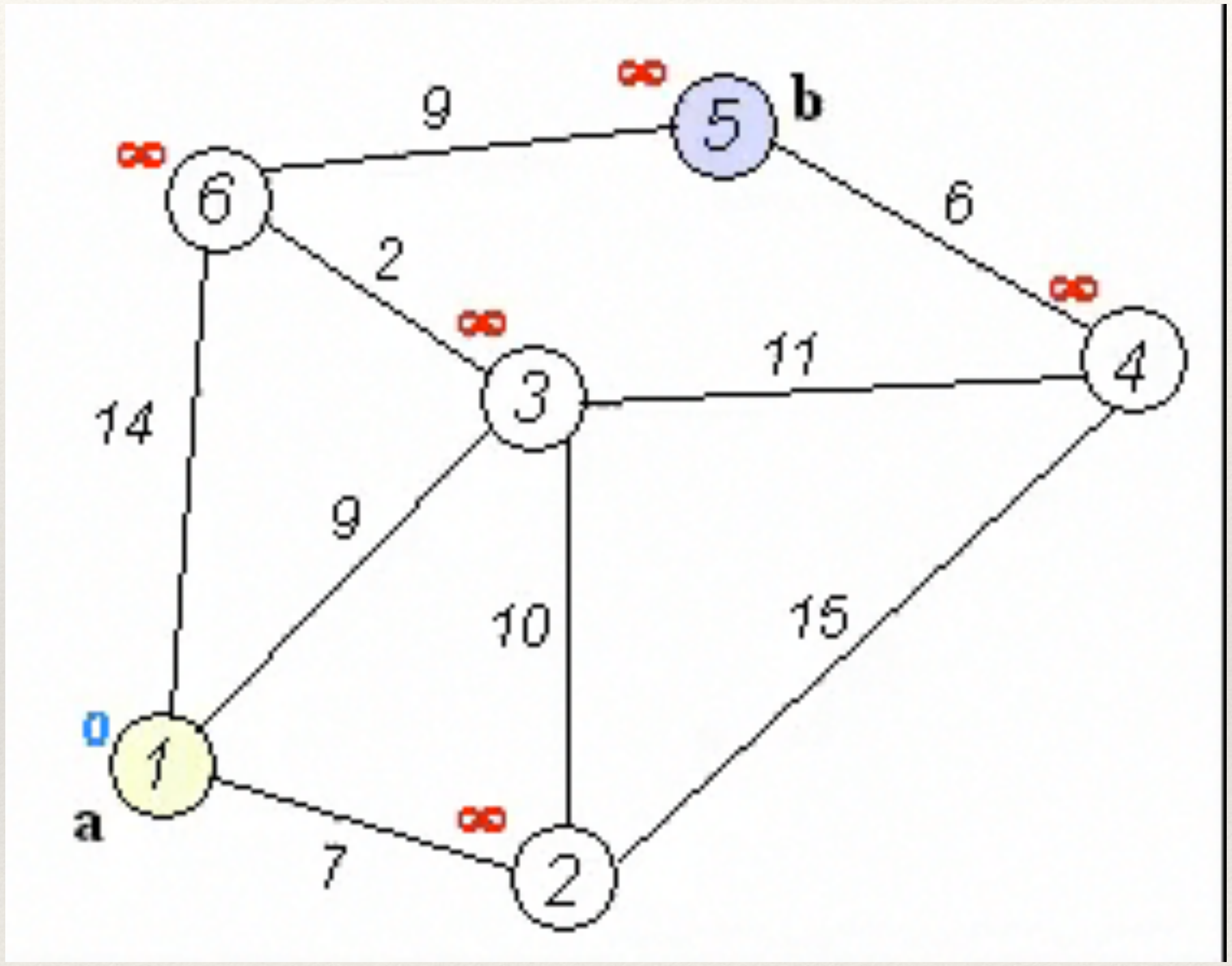
Successive Shortest Paths Min-Cost Flow

- These run times are okay, but still not good enough…
  - The algorithm only requires a single-source shortest path, but we were calculating all-pairs.
  - IDEA! Implement a faster, single-source shortest path algorithm (Dijkstra's)

Dijkstra's Algorithm

- Begin with the starting node, and examine its neighbors; assigning distances according to the edge weights connecting them, then add them to a priority queue, (with priority equal to their current shortest path distance).
- Grab the least cost vertex from the queue, and examine its neighbors, only assigning distances to them if they beat the currently recorded best distance to said neighbor.

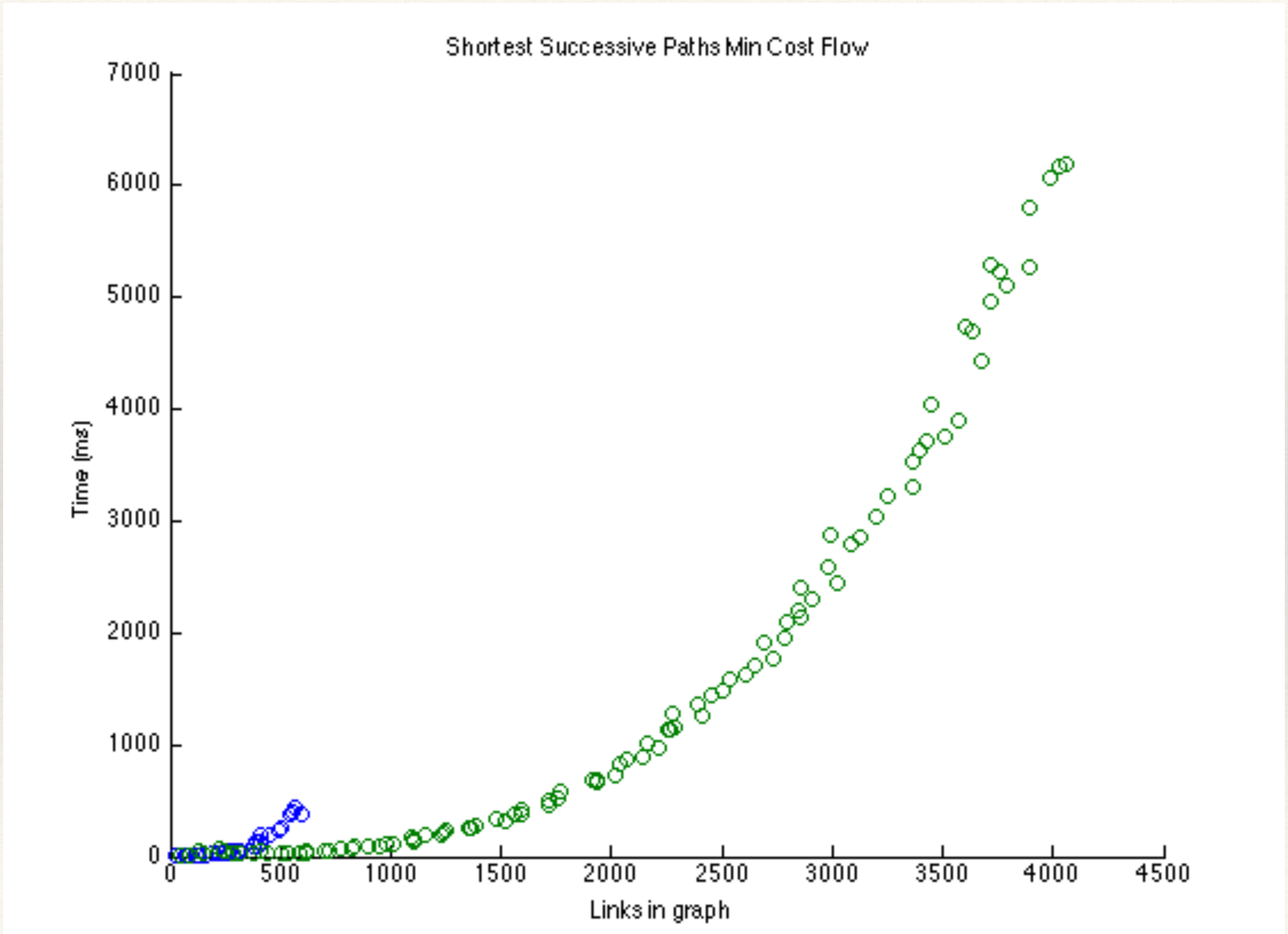
Dijkstra's Algorithm



Dijkstra's Algorithm initialization.

http://en.wikipedia.org/wiki/Dijkstra's_algorithm

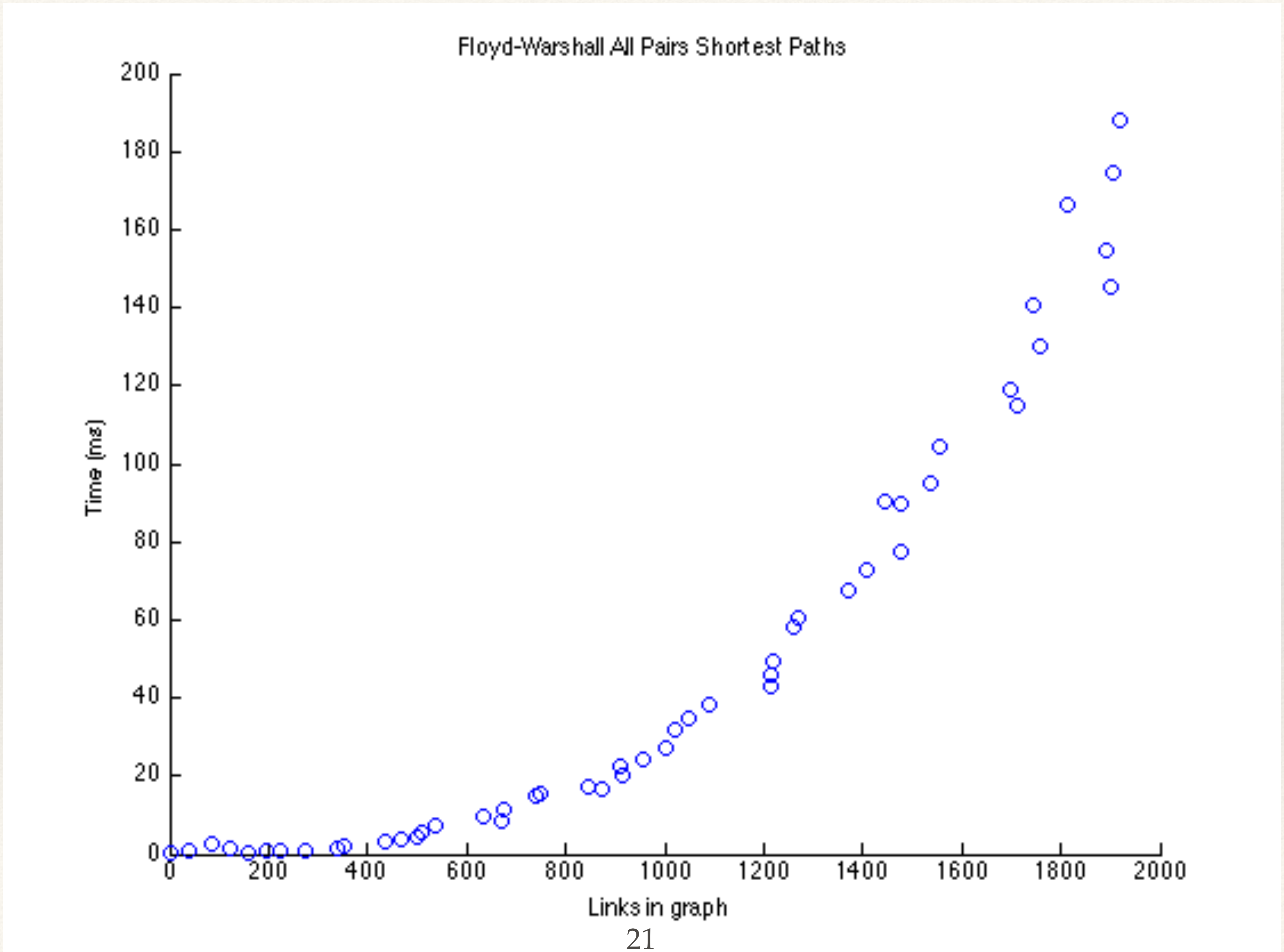
Successive Shortest Paths Min-Cost Flow (With Dijkstras)



Floyd-Warshall All-Pairs Shortest Paths

- Initialize all shortest distances to ∞
- For each pair of vertices between which there exist links, set the shortest distance between them to be the cost of the link (the minimum cost if there are multiple links).
- Then, for a vertex double (**u**,**v**), iterate through the vertices, and see if going from **u** to **w** and then from **w** to **v** is cheaper than our current best.

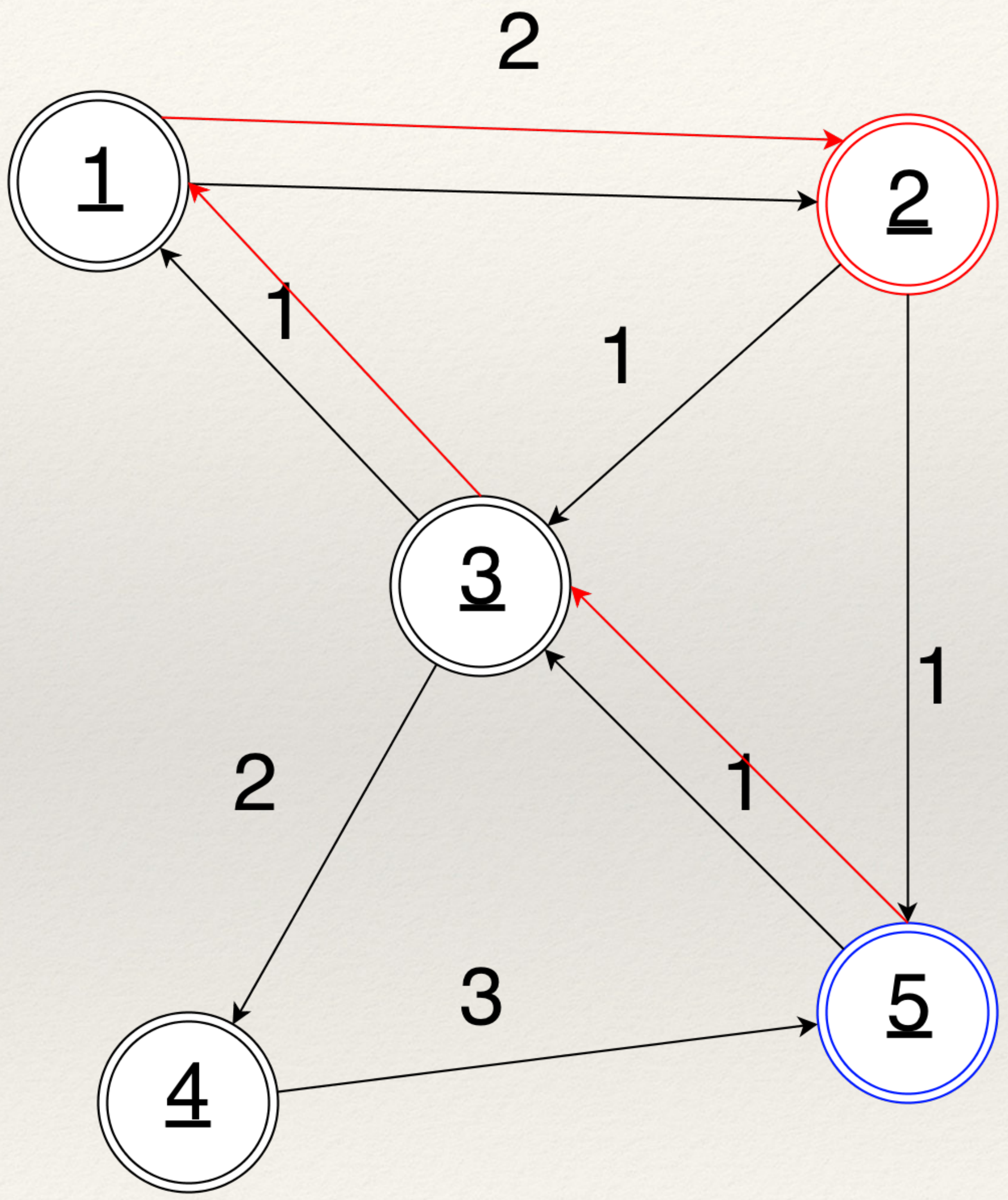
Floyd-Warshall All-Pairs Shortest Paths



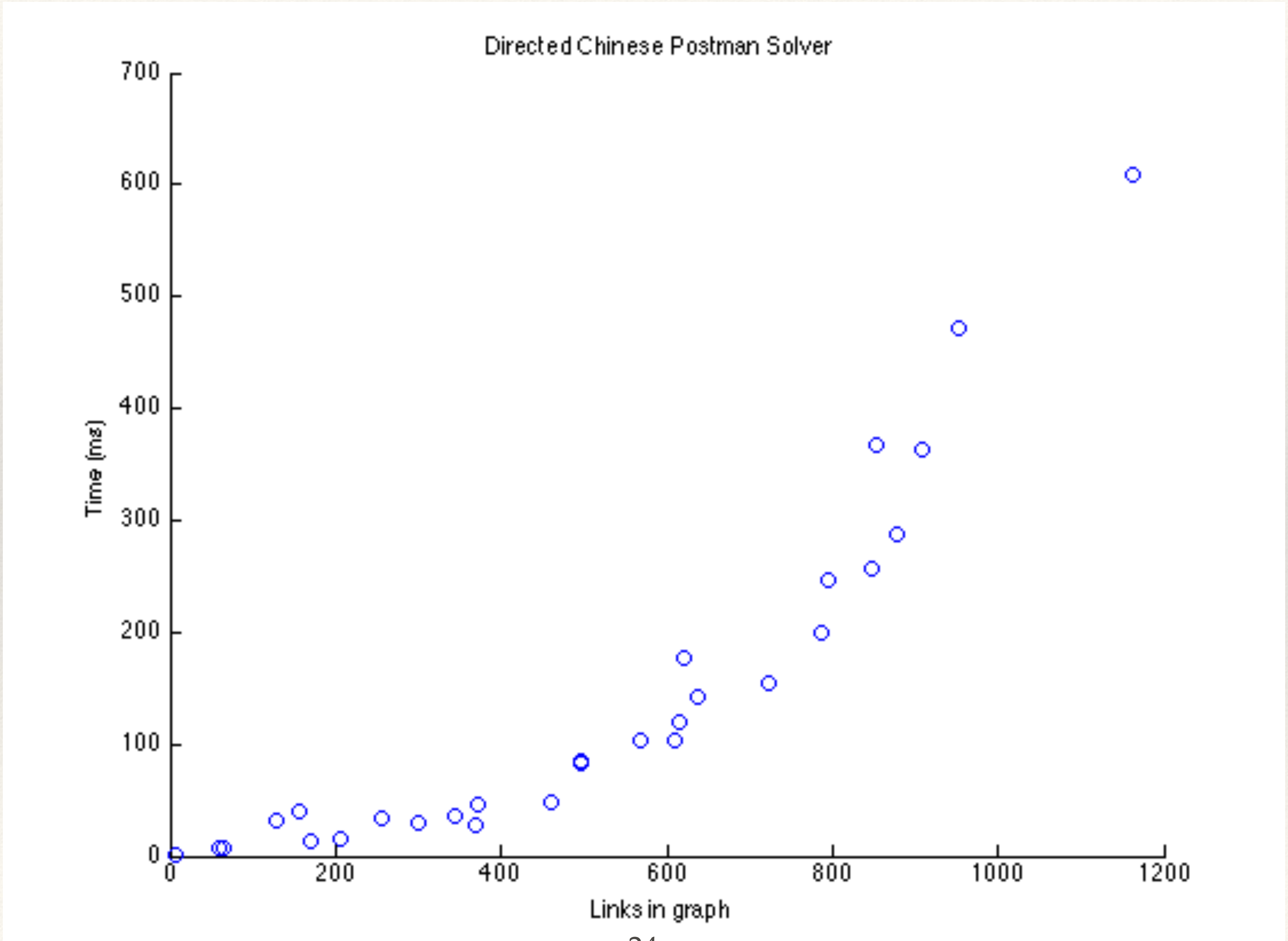
The Directed Chinese Postman Solver

- Eulerian means in-degree = out-degree for all vertices.
- Identify unbalanced vertices, and solve a min-cost flow problem over the graph, where a vertex's supply is given by in-degree - out-degree. Then, for each unit of flow along an arc in the solution, add a copy of it.

The Directed Chinese Postman Solver



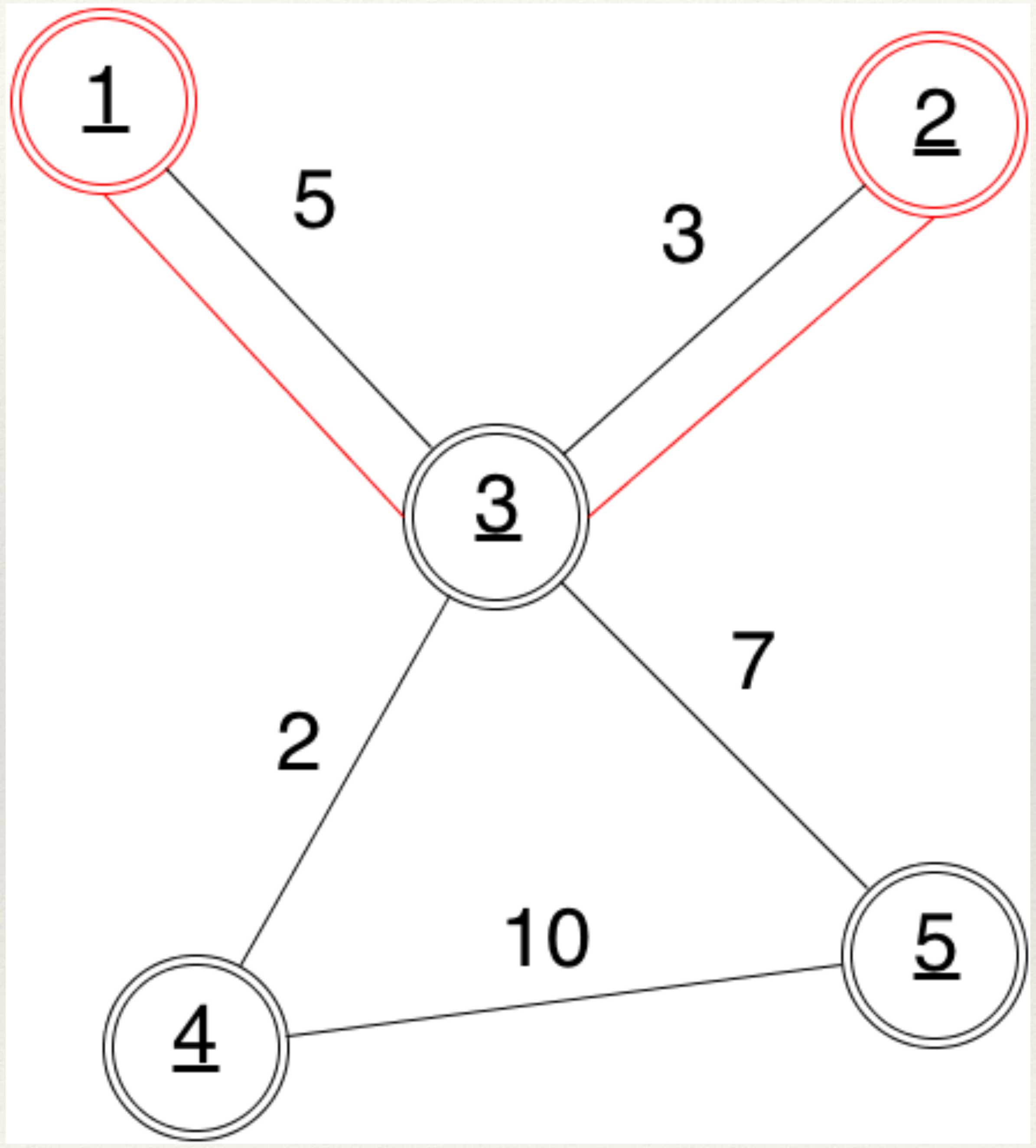
The Directed Chinese Postman Solver



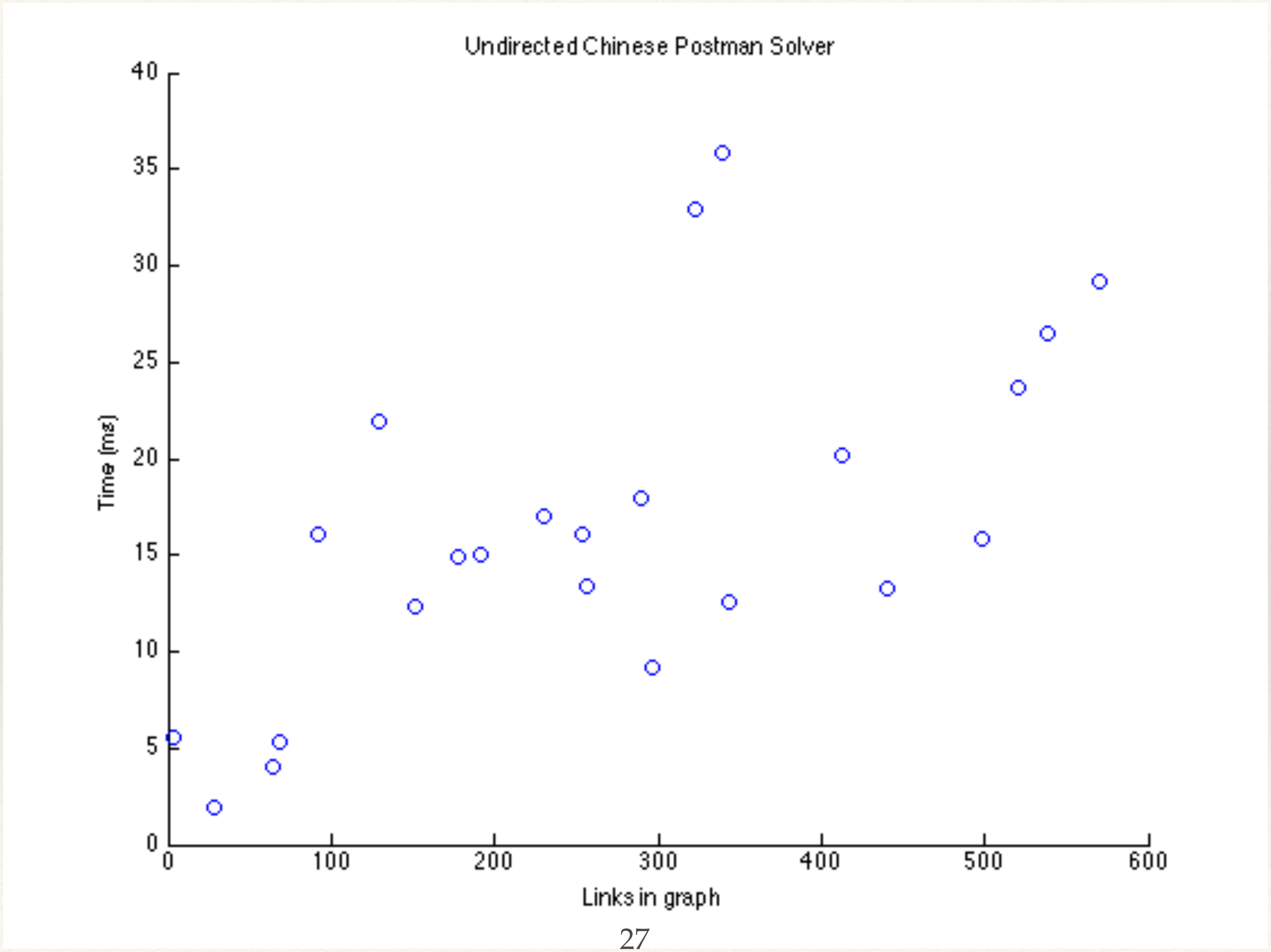
The Undirected Chinese Postman Solver

- Eulerian means degree is even for all vertices.
- Identify the vertices with odd degree, and a solve a min-cost matching on the complete graph of unbalanced vertices, where edge costs are given by shortest path costs in the original. Then add a shortest path between each pair in the matching.

The Undirected Chinese Postman Solver



The Undirected Chinese Postman Solver



Validation

- For the subroutines that we have coded (both shortest path algorithms, and both min-cost flow implementations), we have validated against solvers available in "A Java Library of Graph Algorithms" [11].
- For the Undirected and Directed Chinese Postmen Problems, we have validated them against Gurobi Solvers that we have written which work on the Integer Programming Formulations to the problem.

Validation (UCPP)

$$\text{minimize} \quad \sum_{e \in E} t_e \cdot c_e \qquad \text{subject to} \tag{1}$$

$$\sum_{e \in \delta(v)} (t_e + 1) \equiv 0 \mod 2 \quad \text{for } v \in V \tag{2}$$

$$t_e \geq 0, \text{ integer for } e \in E, \tag{3}$$

Figure from [12]

- Cost function: C
- Edge set: E
- Vertex set: V
- t_e : represents number of additional times we traverse edge e.
- $\delta(v)$: set of edges incident on v.

Validation (DCPP)

$$
\begin{aligned}
&\textbf{determine from the graph}: && \delta, D^-, D^+, c \\
&\textbf{find } f \textbf{ to minimize}: && \sum c_{ij} f_{ij} \\
&\textbf{subject to}: && \begin{cases} f_{ij} \text{ integer} \\ f_{ij} \geq 0 \\ \displaystyle\sum_{j \in D^+} f_{ij} = -\delta(i) \\ \displaystyle\sum_{i \in D^-} f_{ij} = \delta(j) \end{cases}
\end{aligned}
$$

Figure from [1]

- Cost function: C
- f_{ij} : represents number of additional times we traverse the shortest path from i to j.
- $\delta(v)$: outdegree – indegree of vertex v.
- $D\uparrow+$: set of vertices with excess outgoing arcs.
- $D\uparrow-$: set of vertices with excess incoming arcs.

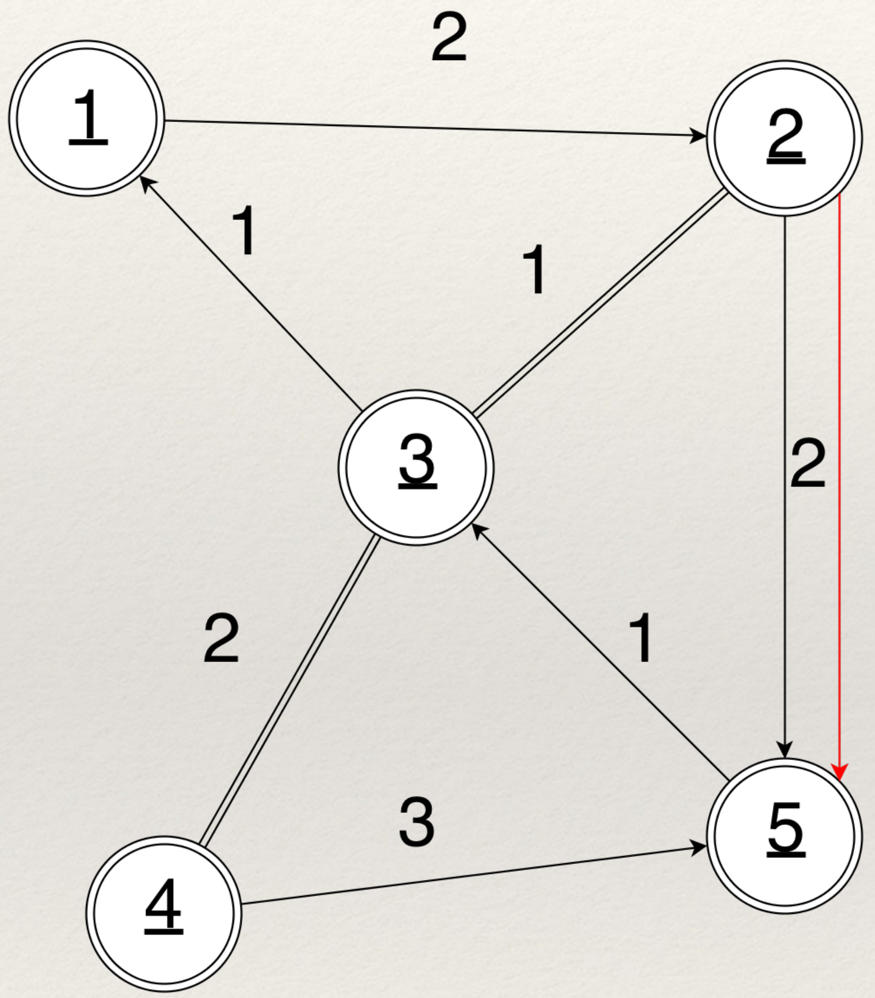
A Key Comparison

- How much better is this method than the IP Formulation against which we validate?
  - Depending on the problem, very much so:
    - On a graph with ~300 links, solve times for Gurobi regularly varied between 10 ms, and 1200 ms.
    - Gurobi is an industrial grade solver, written in a low level language, with highly optimized code: designed to be performant! (Sub-text, mine is not that good)

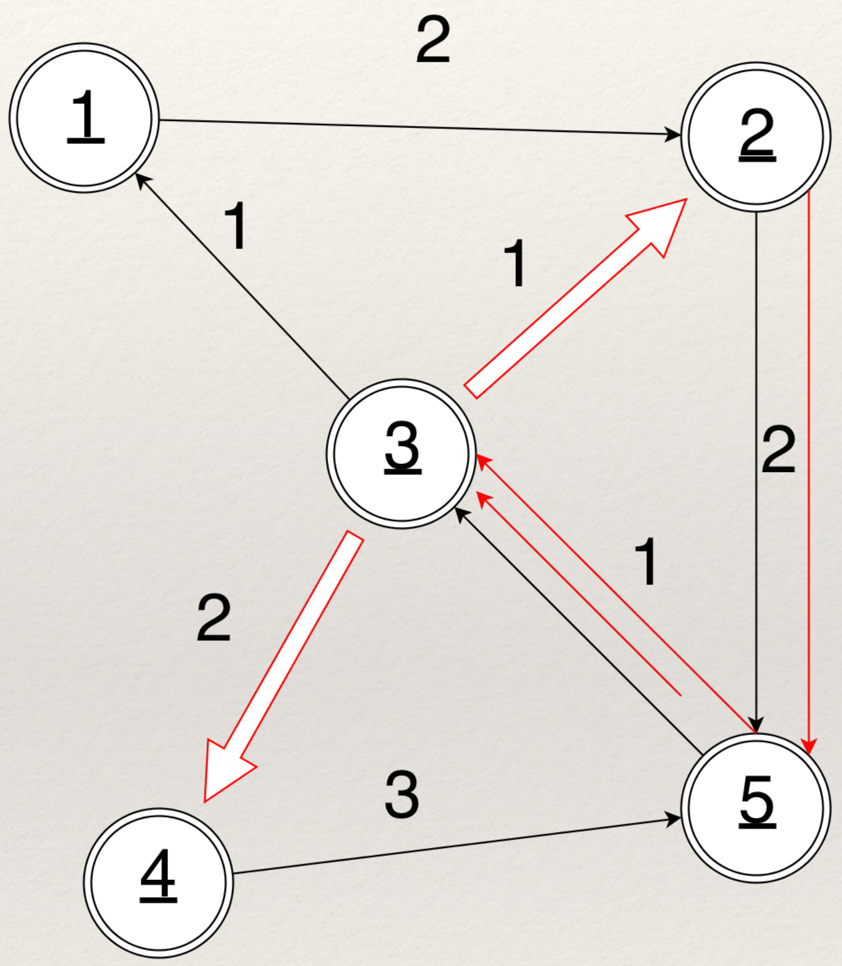
Frederickson's Mixed-1

- Two-Stage Heuristic: Choose the better answer from the results of 2 different heuristics.
  - Mixed1:
    - Even Degree: Suppose all arcs were edges; solve the UCPP. (Add copies of arcs as arcs).
    - In-Out Degree: Solve the following min-cost flow problem:
      - Demands / supplies given just as in the DCPP
      - Each arc in the original graph appears in the flow network with identical cost / direction.
      - Each edge in the original appears as 4 arcs in the flow network:
        - One in each direction with cost equal to the edge cost, and capacity ∞.
        - One in each direction with zero cost, and capacity 1.
    - For each unit of flow across a link in the solution, add a copy of it to the original graph.
    - Even Parity: Find cycles that may be eliminated to restore evenness.
      - Determine the odd-degree vertices.
      - Construct two graphs: one with only edges left undirected thus far; one with only arcs added during In-Out Degree.
      - Greedily travel from odd vertex to odd vertex, alternating which graph you look in. Orient the edges traversed, and add / delete copies (depending on direction of traversal) of the arcs traversed.

Frederickson's Mixed Chinese Postman Algorithm

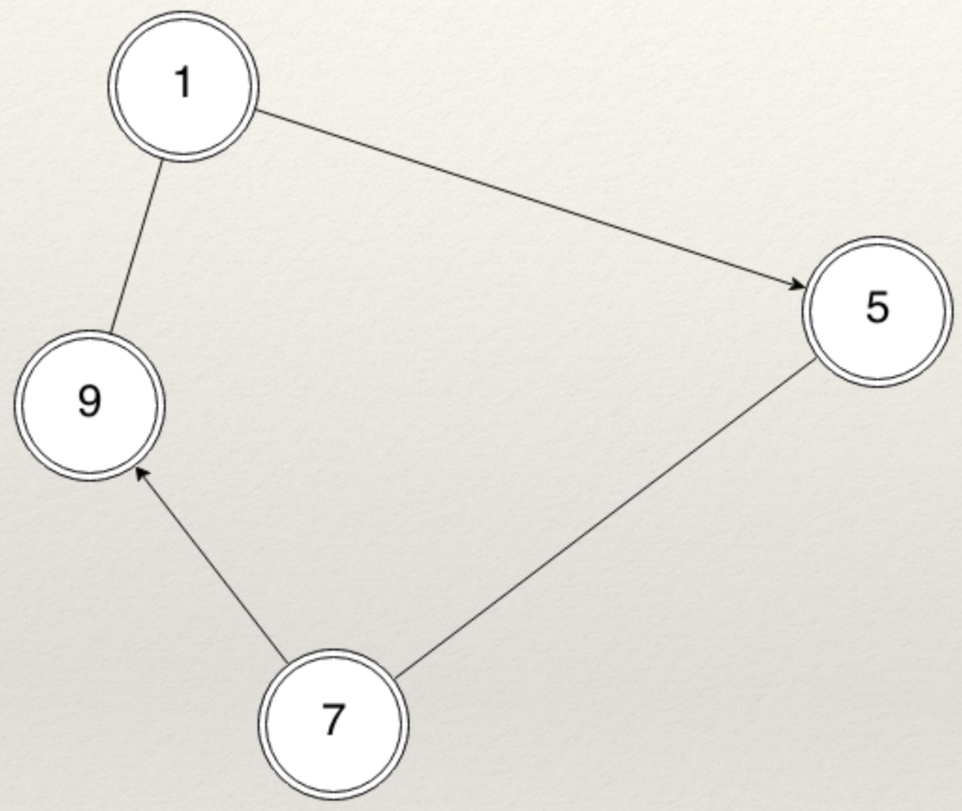


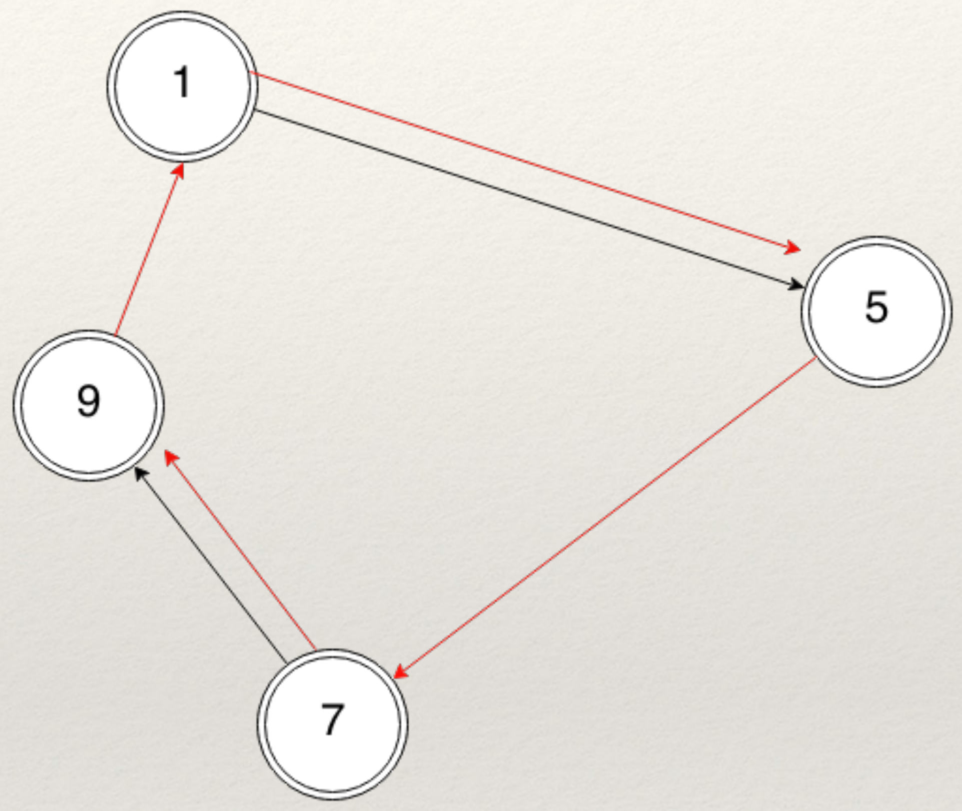
(Even Degree)



(In-Out Degree)

Frederickson's Mixed Chinese Postman Algorithm



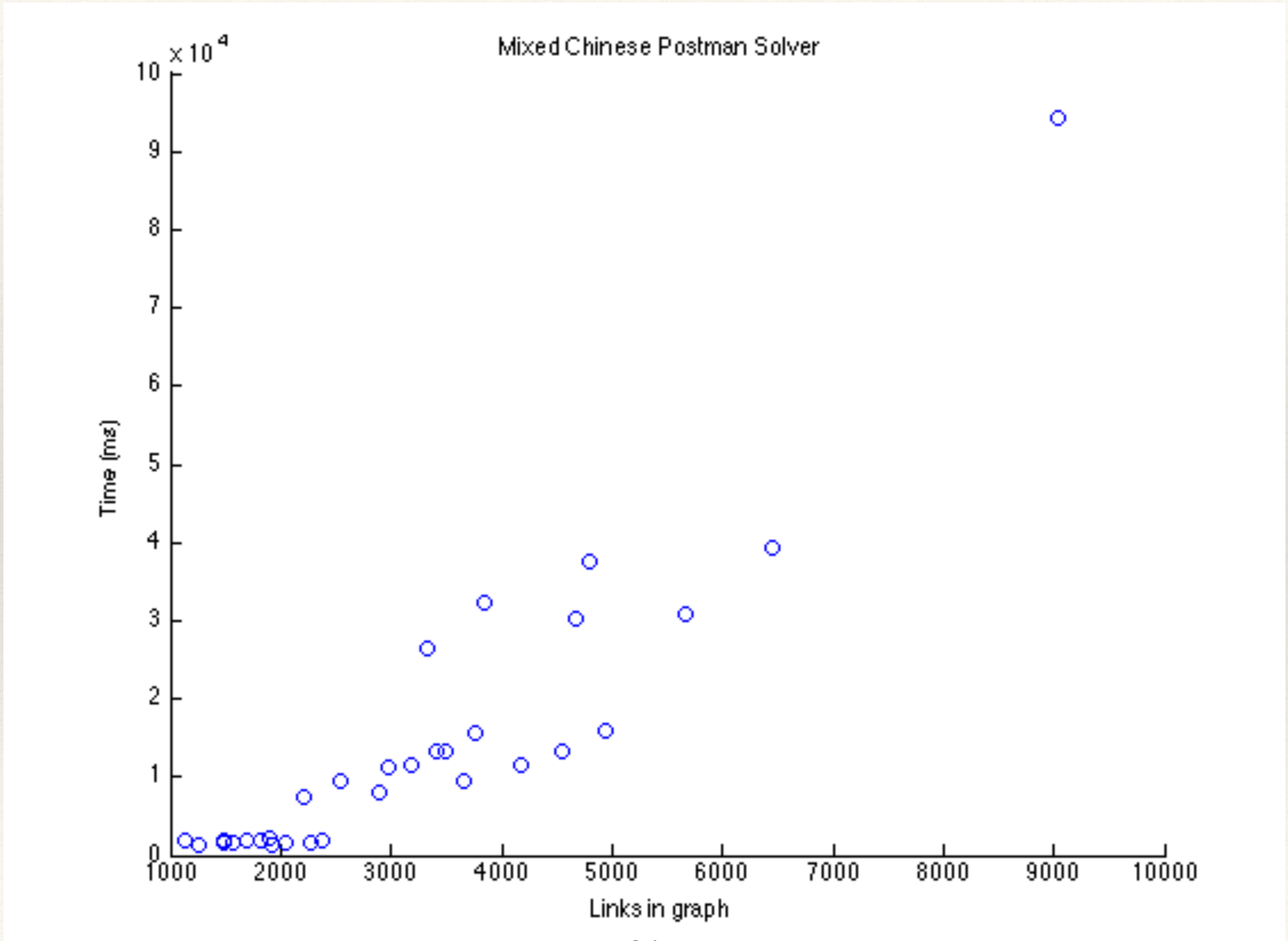


(Even Parity)

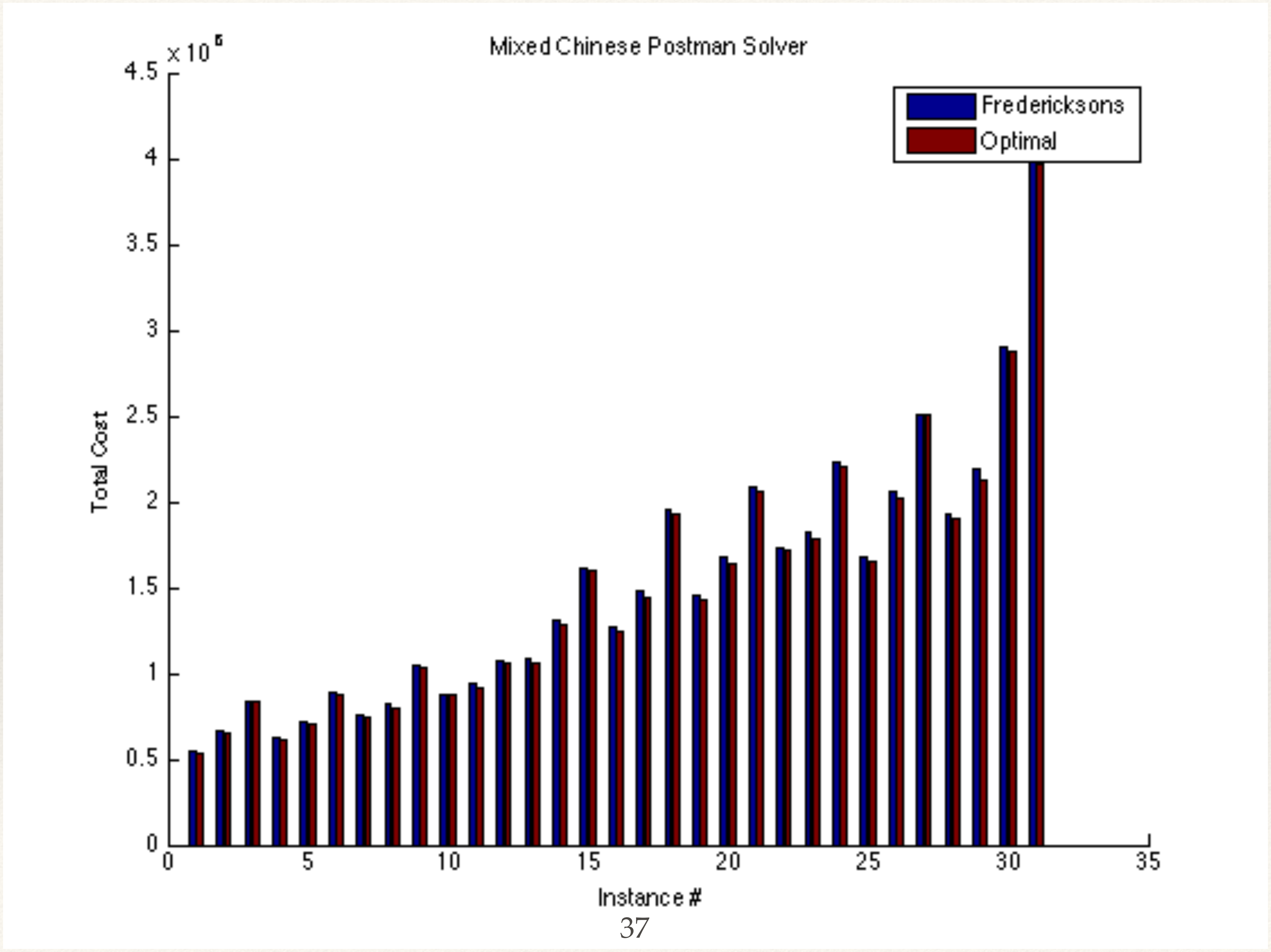
Frederickson's Mixed-2

- Mixed2:
    - In-Out Degree: (Same as previous)
    - Large Cycles: Solve a min-cost matching among the odd vertices, (where odd vertices are found after in-out degree has 'directed' some edges.

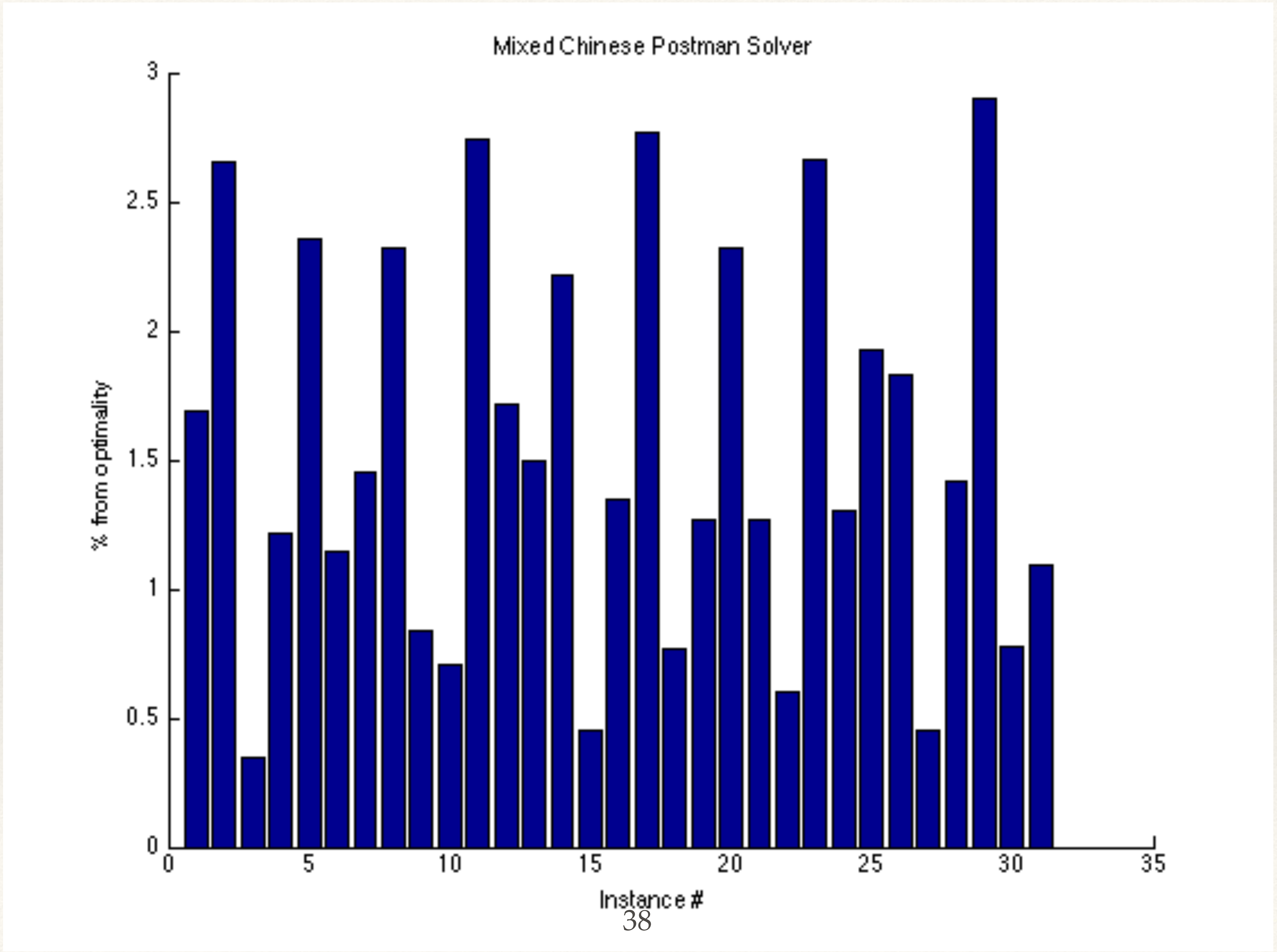
Frederickson's Mixed Chinese Postman Algorithm



Frederickson's Mixed Chinese Postman Algorithm

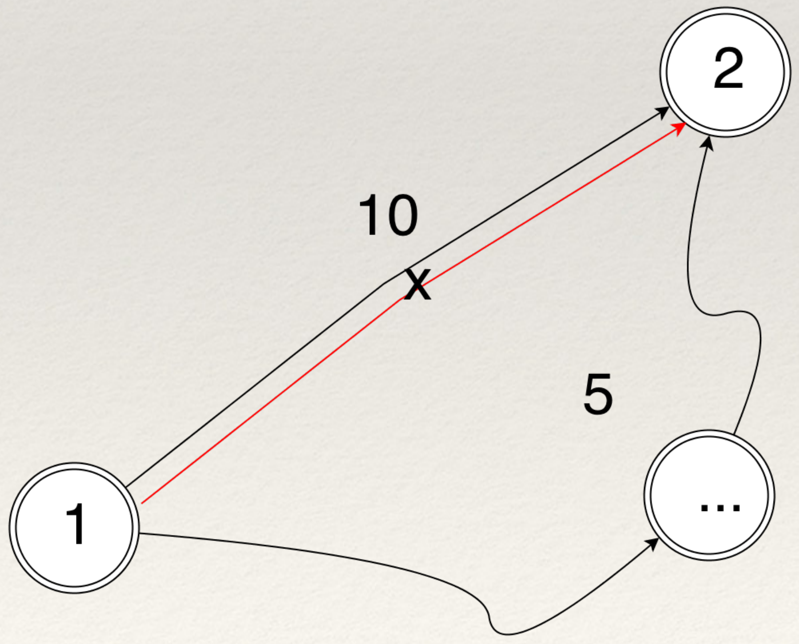


Frederickson's Mixed Chinese Postman Algorithm



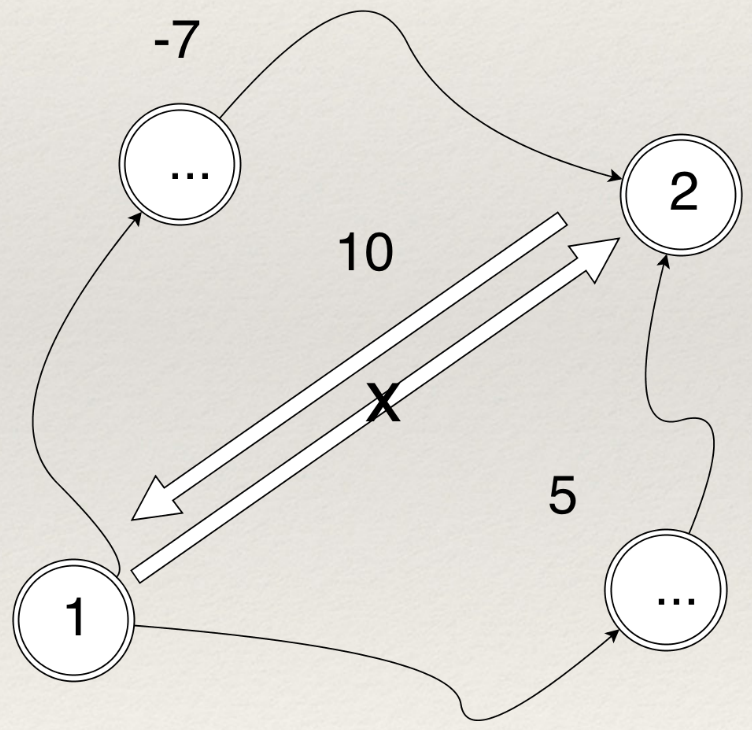
The Shortest Additional Paths Heuristic (MCPP)

- Builds on work done by Frederickson's:
  - In-Out Degree: (Same as previous)
  - SAPH Concept #1: For an added link <i - j>, see if we can do better by adding a shortest path from i to j.



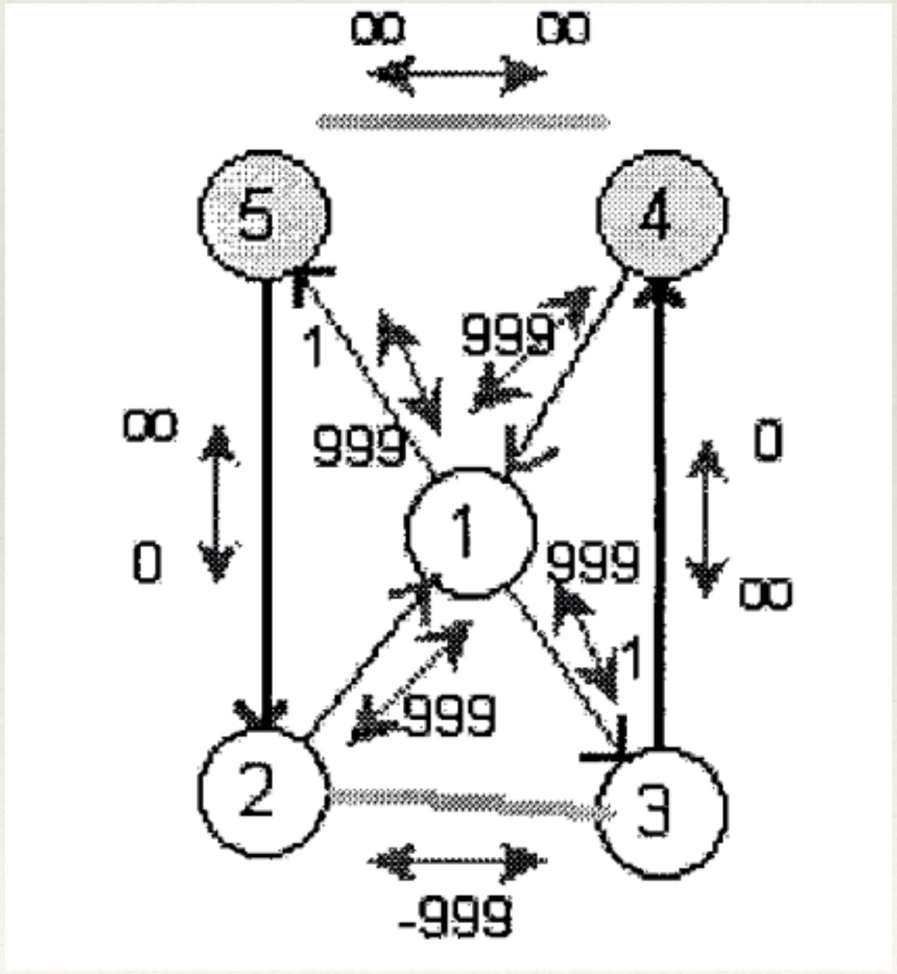
The Shortest Additional Paths Heuristic (MCPP)

- SAPH Concept #2: For a ‘directed’ edge (i —> j), see if we can do better by adding the two shortest paths from i to j, and reversing direction



The Shortest Additional Paths Heuristic (MCPP)

- Oh no! No way to solve for shortest paths in a graph with negative cycles! (correspondence with author under way).



Schedule

Old

- DCPP Solver (Complete, and Validated)
- UCPP Solver (Complete, and Validated)
- Test Framework (Complete)
- MCPP Heuristics (1/2)
  - Frederickson's (Complete)
  - Shortest Additional Paths (?)
- WPP Heuristics
  - Win's - (December)
  - Benavent's - (December)
- DRPP Heuristics
  - Christofides' - (January)
  - Benavent's - (January)
- Performance Optimization** (February)
- Gurobi Integration** (March)
- Visualization** (April)
- Final Report (May)

Revised

- DCPP Solver (Complete, and Validated)
- UCPP Solver (Complete, and Validated)
- Test Framework (Complete)
- MCPP Heuristics (1/2)
  - Frederickson's (Complete)
  - Shortest Additional Paths (?)
- WPP Heuristics
  - Win's - (December / January)
  - Benavent's - (December / January)
- DRPP Heuristics
  - Christofides' - (February / March)
  - Benavent's - (February / March)
- Performance Optimization** (Ongoing)
- Gurobi Integration** (Partially Complete)
- Visualization** (April)
- Final Report (May)

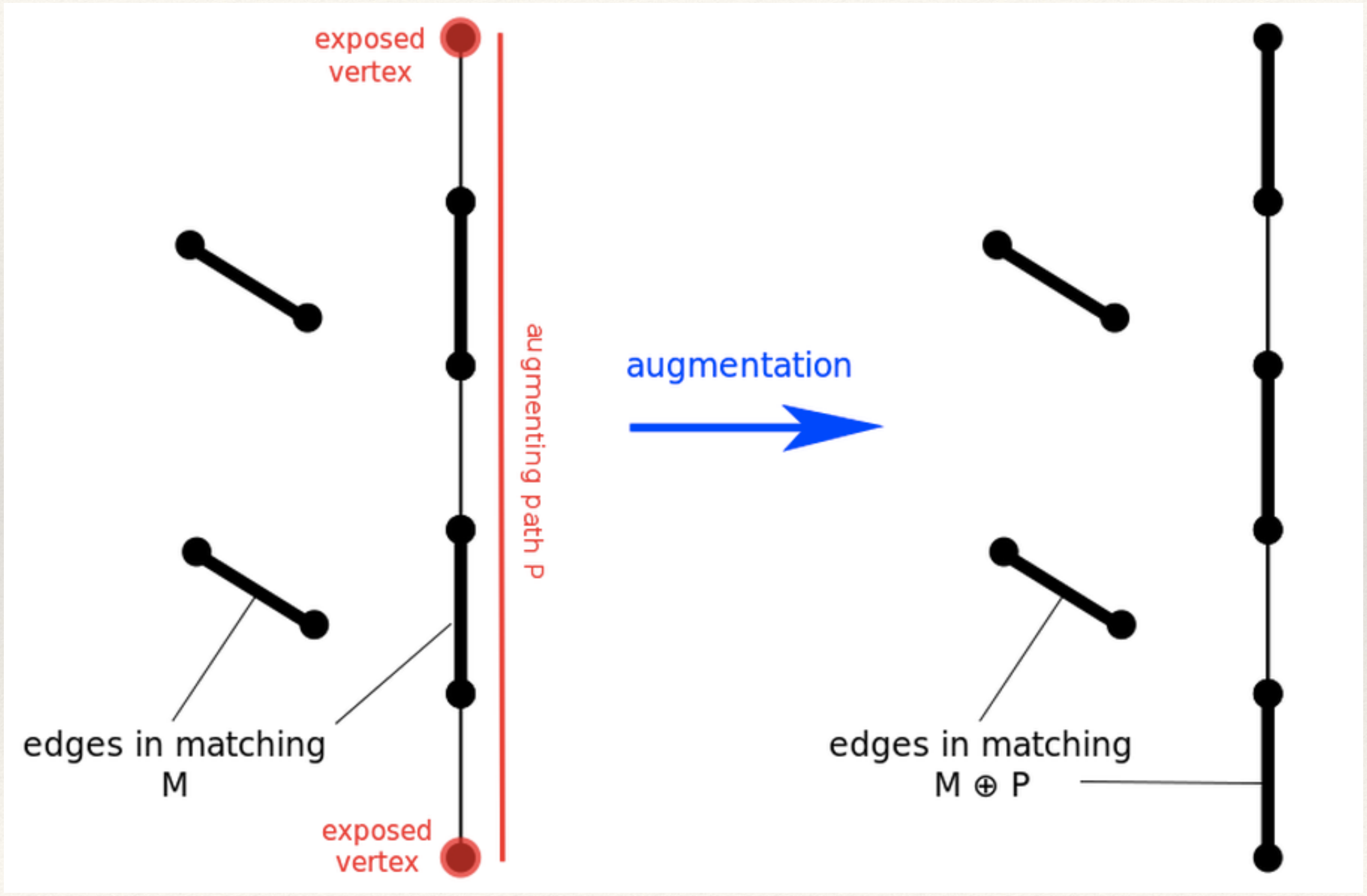
References

1. Thimbleby, Harold. "The directed chinese postman problem." *Software: Practice and Experience* 33.11 (2003): 1081-1096.
2. http://www.ise.ncsu.edu/fangroup/or766.dir/or766_ch9.pdf
3. Eiselt, Horst A., Michel Gendreau, and Gilbert Laporte. "Arc routing problems, part I: The Chinese postman problem." *Operations Research* 43.2 (1995): 231-242.
4. Yaoyuenyong, Kriangchai, Peerayuth Charnsethikul, and Vira Chankong. "A heuristic algorithm for the mixed Chinese postman problem." *Optimization and Engineering* 3.2 (2002): 157-187.
5. Benavent, Enrique, et al. "New heuristic algorithms for the windy rural postman problem." *Computers & operations research* 32.12 (2005): 3111-3128.
6. Eiselt, Horst A., Michel Gendreau, and Gilbert Laporte. "Arc routing problems, part II: The rural postman problem." *Operations Research* 43.3 (1995): 399-414.
7. Campos, V., and J. V. Savall. "A computational study of several heuristics for the DRPP." *Computational Optimization and Applications* 4.1 (1995): 67-77. (Replace this with Carmine's paper when I get it).
8. Dussault, Benjamin, et al. "Plowing with precedence: A variant of the windy postman problem." *Computers & Operations Research* (2012).
9. Win, Zaw. "On the windy postman problem on Eulerian graphs." *Mathematical Programming* 44.1-3 (1989): 97-112.
10. Christofides, Nicos, et al. "An algorithm for the rural postman problem on a directed graph." *Netflow at Pisa*. Springer Berlin Heidelberg, 1986. 155-166.
11. Lau, Hang T. A Java library of graph algorithms and optimization. CRC Press, 2010.
12. Derigs, Ulrich. Optimization and operations research. Eolss Publishers Company Limited, 2009.

BONUS SLIDE!!! Min-Cost Perfect Matching

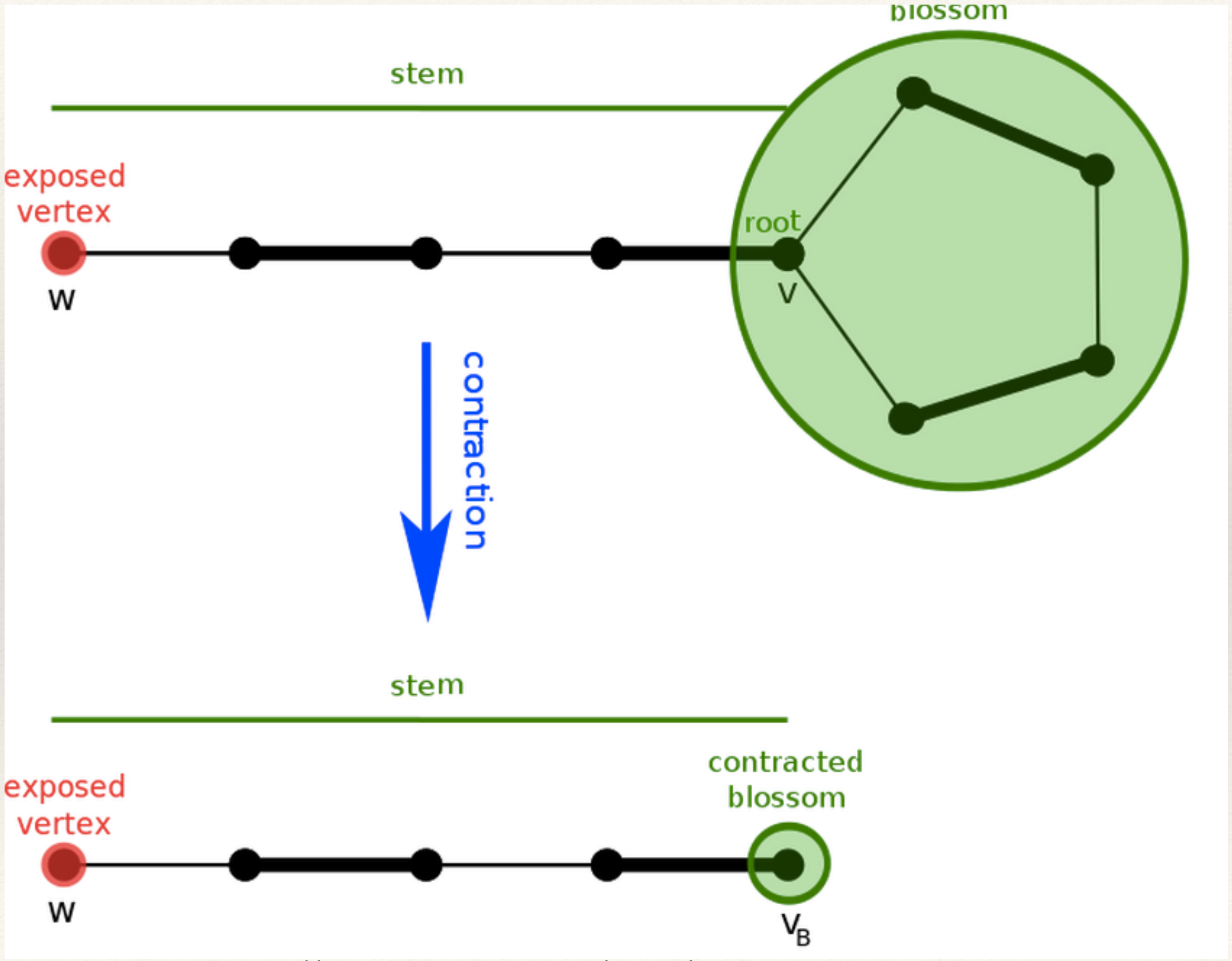
- Edmond’s Blossom Algorithm:
  - Based off an algorithm to find a maximum matching.
  - Operates on the dual of an LP set up to determine the min-cost matching.
  - At each iteration, only augmenting paths with zero reduced cost are candidates for addition.
  - Terminates when no more augmenting paths.

BONUS SLIDE!!! Min-Cost Perfect Matching



http://en.wikipedia.org/wiki/Blossom_algorithm

BONUS SLIDE!!! Min-Cost Perfect Matching



http://en.wikipedia.org/wiki/Blossom_algorithm