Imperial College London

15 December 2024

Network connectivity augmentation

Waner Chen, Thomas Pink (CID: 01854827, 06003339)

Imperial College London

Key Results and Complexity

Problem Statement: Given a weakly connected network G, determine the smallest number of links to make G strongly connected.

Key Result:

$$\text{Minimum links to add} = \max(\text{source}(C), \text{sink}(C))$$

where C is the **condensation** of the network G.

Algorithmic Complexity:

$$O(V + E + H^2)$$

where V is the number of vertices, and E is the number of edges in G and H the number of sinks of the condensation C.

Imperial College London

Applications

Algorithm Applications:
We consider both non-financial and financial applications.

Imperial College London

Wikipedia Game

Problem:
Navigate between any two pages in Wikipedia using hyperlinks.

Challenge:
Wikipedia is not strongly connected, which prevents navigation between some pages.

Goal:
Add the minimal number of links (edges) to ensure strong connectivity for the entire Wikipedia graph, making navigation seamless for users.

Imperial College London

Spy Ring

Scenario:
A spymaster's communication network between agents becomes compromised, leaving agents disconnected.

Challenge:
Reestablish secure connectivity while minimizing the number of links added to avoid enemy detection.

Goal:
Add the fewest links to make the network strongly connected, thus ensuring all agents can securely send and receive messages.

Imperial College London

Settlement Networks

Scenario:

A payment network processes transactions through participants and exchanges. Failures or downtime may lead to disconnected networks.

Challenge:

Prevent or fix a disconnection by adding a minimum number of new (possibly one way) links between exchanges to allow full communication in the network

Goal:

Add minimal edges to ensure the graph remains strongly connected, enabling a robust settlement network.

Imperial College London

High-Frequency Trading Networks

Scenario:

High-frequency traders use specialized communication networks to reduce latency between exchanges. Failures in these networks can lead to significant disruptions.

Challenge:

Reroute communication during failures while minimizing the number of edges added.

Goal:

Identify and add the necessary edges to ensure the network is resilient and capable of maintaining low-latency communication between exchanges.

Imperial College London

Payment Settlement Matching

Scenario:
A series of institutions have made a series of transactions between each other, that don't fully net off.

Challenge:
Encourage participants to make certain transactions to allow for netting off.

Goal:
As participants may be reluctant to trade, identify the minimum number of transactions required for a netting off cascade to be possible.

Imperial College London

Key Definitions

Definitions

On the following slides we define key concepts used in the proof.

Graph

Definition:

A graph $G(V, E)$ is a set of vertices V (also referred to as nodes), and a set of pairs of vertices E, with elements (u, v) with $u, v \in V$, referred to as edges.

A directed graph has the elements of E being ordered, i.e., $(u, v) \neq (v, u)$, whereas an undirected graph has the elements of E unordered.

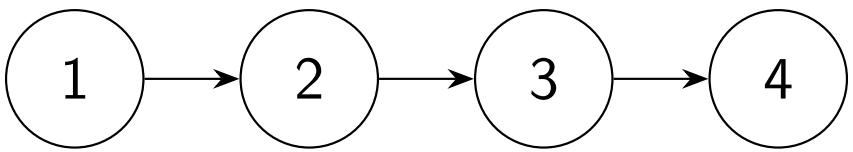


Figure: A simple directed graph with vertices 1, 2, 3, and 4 in a single line.

Strongly/Weakly Connected

Definition:

A directed graph is **strongly connected** if there is a directed path from any vertex to any other.

A graph is **weakly connected** if there is a path joining any pair of vertices without considering the direction of edges.

Example Use Cases:

- Strongly connected graphs are used in routing and communication systems.
- Weakly connected graphs are used in social networks and loosely coupled systems, in payment systems, one way road networks or flow networks.

Sinks, Sources, and Transient Vertices

Definition:

A vertex v in a directed graph G is a **source** if $v_{\text{indegree}} = 0$, and is a **sink** if $v_{\text{outdegree}} = 0$.

A vertex is called **transient** if it is neither a sink nor a source.

Notation Summary:

- SINK(G): Total number of sinks in graph G.
- SOURCE(G): Total number of sources in graph G.
- H: Set of sink vertices.
- S: Set of source vertices.

Component

Definition:
A component $C(U, F)$ of a graph $G(V, E)$ is a set of vertices U, where $U \subseteq V$, and all associated edges $F \subseteq E$ between vertices of U in G.

If a component of a directed graph is strongly/weakly connected, then we say it is **strongly/weakly connected**, respectively.

A component is **trivial** if it is a single vertex.

Imperial College London

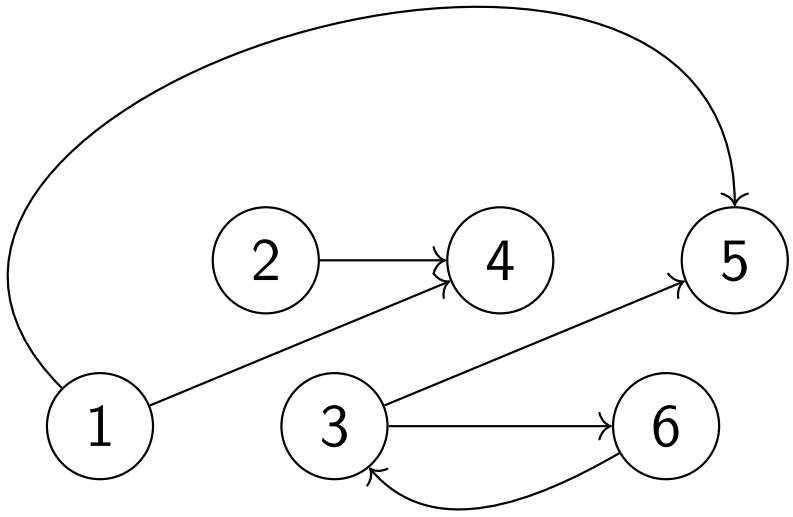
Condensation

Definition:
A **condensation** $R(U, F)$ of a directed graph $G(V, E)$ is the graph obtained by replacing each maximal strongly connected component C with a vertex c, such that for each $c \in U$, we have that for $d \in U$, $c \neq d$, which represent the maximal strongly connected components (C, D) with $(C \neq D)$ respectively:

$$(c, d) \in F \iff \exists u, v \in C, D \text{ such that } (u, v) \in E.$$

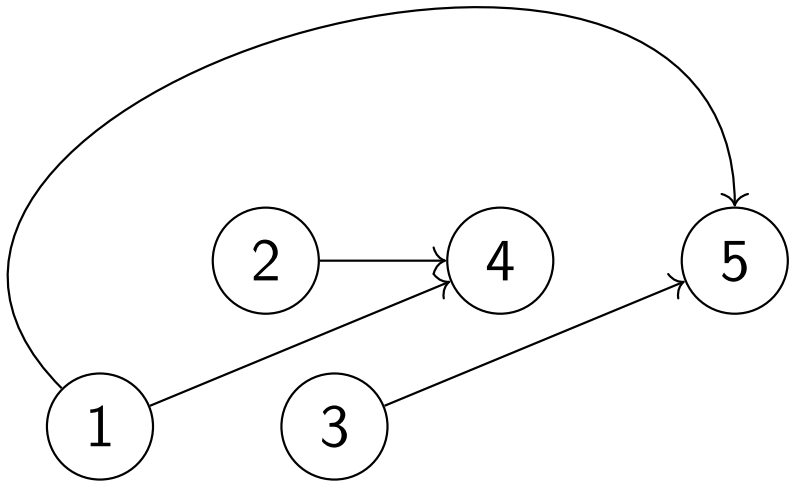
Imperial College London

Condensation Example Pre Condensed



Imperial College
London

Condensation Example Condensed



Acyclic

Definition:
A directed graph G is **acyclic** if there are no paths from a vertex to itself.

Note that condensations must be acylic, since a cycle is a strongly connected component.

Imperial College London

Vision

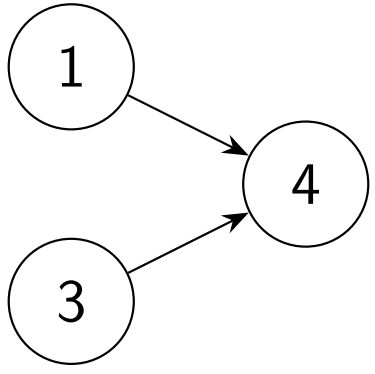
Definition:

The **vision**, $Vis(v)$, of a vertex v in an acyclic graph G is the set of sinks that can be reached from that vertex.

We say that v can see an element $h \in Vis(v)$.

Example:

In the graph below, the vision of vertex 3 is $\{4\}$.



Lemmas for the Proof

Lemmas:
On the following slides we note some key lemmas that make the proof conceptually easier to understand.

Lemma 1

Statement:

Let C be the condensation of G. Then:

C is strongly/weakly connected $\iff$ G is strongly/weakly connected.

Imperial College London

Lemma 2

Statement:

For a directed graph G with condensation C, denoting a vertex $v \in G$ that is sent to the corresponding vertex $c_v \in C$ in the natural way:

- Adding an edge (u, v) to G and then condensing.
- Adding an edge (c_u, c_v) in C and then condensing.

Obtains the same condensed graph.

Lemmas 1 and 2

Comment:
Lemma 1 and Lemma 2 show that adding edges to the condensed graph and then condensing repeatedly is equivalent to adding edges to the original.

Crucially if during this process the graph becomes strongly connected, then we could strongly connect G with the same number of edges.

Lemma 3

Lemma:
Let C be the condensation of a directed graph G. To make C strongly connected, we must add at least:

$$\max(\text{SINK}(C), \text{SOURCE}(C))$$

edges.

Imperial College London

Lemma 4

Statement:

Every vertex in a condensed graph C generated from a finite weakly connected graph G can be reached from a source.

Imperial College London

Lemma 4 Corollary

Statement:

The union of the visions of the sources is the set of sinks H

Imperial College
London

Theorem and Statement

Theorem:
We now state the theorem, and prove it using the lemmas.

Minimum Edges for Strong Connectivity 1

Theorem:
Let G be a directed graph that is not strongly connected, with finite condensation C. Then G can be strongly connected with:

$$\max(\text{SINK}(C), \text{SOURCE}(C)),$$

edges, and this is the minimum number of edges required.

Imperial College London

Minimum Edges for Strong Connectivity 2

Opening Key Points:

1. We can repeatedly condense, and strongly connect the condensations to strongly connect the original graph.
2. Each time we add an edge, we need to be careful that we have reduced the number of sources and sinks by 1.
3. In each condensation, as transitive vertices are reachable from a source, we need only consider the visions of sources.
4. We know if we achieve the minimum in lemma 3, we cannot do any better.

Imperial College London

Minimum Edges for Strong Connectivity Proof 1

Weakly Connecting the Graph

We assume the graph was weakly connected.

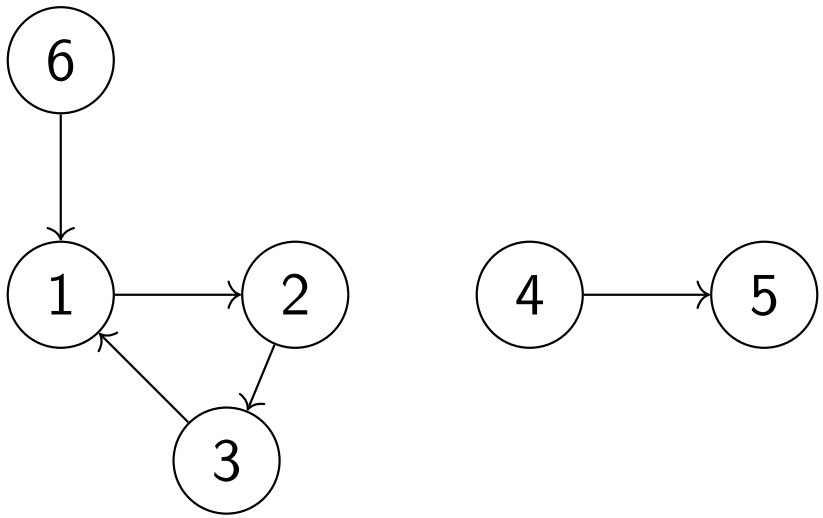
If not, we add edges to the condensation, sink to source in series.

The condensation of this new graph is itself, and so we proceed with this graph.

Imperial College London

Minimum Edges for Strong Connectivity Proof 2

Weakly Connecting the Graph



Imperial College
London

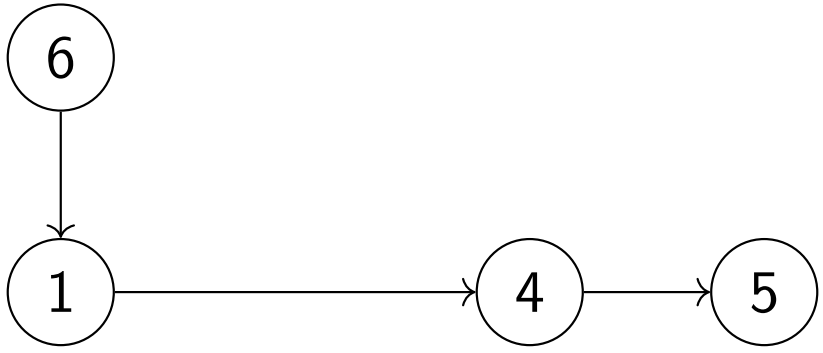
Minimum Edges for Strong Connectivity Proof 3

Weakly Connecting the Graph

Imperial College London

Minimum Edges for Strong Connectivity Proof 4

Weakly Connecting the Graph



Imperial College London

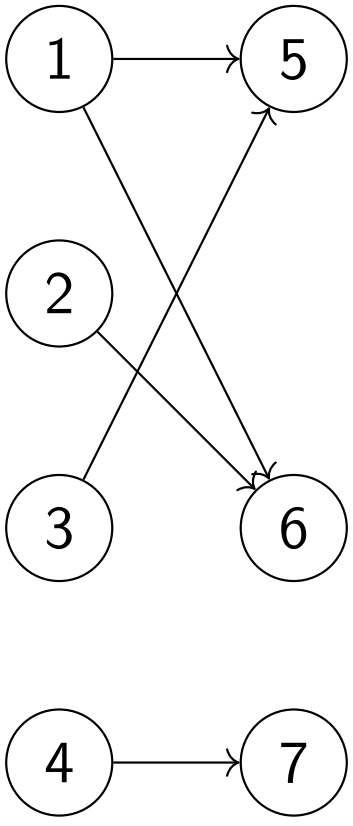
Minimum Edges for Strong Connectivity Proof 5

Steps to prove:

1. We choose a source and connect sinks it can see to other sources until the original source can see all remaining sinks.
2. We then replace the source that can see everything, with a new source than can see everything until one sink or source is left
3. We connect the remaining sinks to the source, or source to the sinks

Imperial College London

Minimum Edges for Strong Connectivity Proof 6



Imperial College London

Minimum Edges for Strong Connectivity Proof 7

Step 1:

We take a source, s . If it's vision is all sinks, we move onto step 2. Otherwise, there is a source σ with a sink, h in its vision that cannot be seen by s. (Since the union of the vision of all sources is the set of sinks)

Connect a sink from $Vis(s)$ to σ and then condense.

Minimum Edges for Strong Connectivity Proof 8

Step 1 cont'd:

Since s can see h in the condensed graph, we could not have created another source or sink.

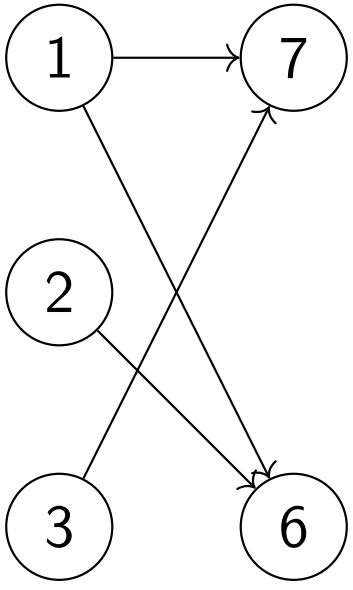
Therefore the number of sources and sinks in the new condensed graph have been reduced by 1 each.

Repeat this process if s cannot see everything, otherwise move to step 2.

Imperial College
London

Minimum Edges for Strong Connectivity Proof 9

Results after step 1



Imperial College London

Minimum Edges for Strong Connectivity Proof 10

Step 2

We must now have a source s that can see all remaining sinks. If it is the only source, or if there is one sink, move onto step 3.

Otherwise, take another source σ.

Connect a sink $h \in Vis(\sigma)$ to s, and condense.

It is clear that σ can see all remaining sinks.

Imperial College London

Minimum Edges for Strong Connectivity Proof 11

Step 2 cont'd

We could not have created a new sink or source, since there is a path from σ to all remaining sinks, passing through the condensed component containing the new edge we added.

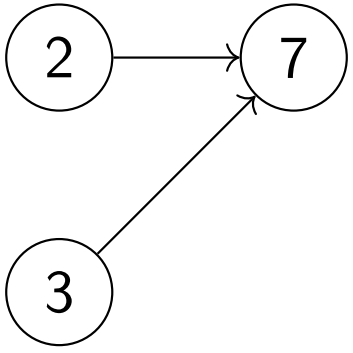
So we have reduced the number of sinks and sources by 1 each.

If there is 1 sink or source remaining, move onto step 3. Otherwise repeat this step.

Imperial College
London

Minimum Edges for Strong Connectivity Proof 12

Results after step 2



Imperial College London

Minimum Edges for Strong Connectivity Proof 13

Step 3

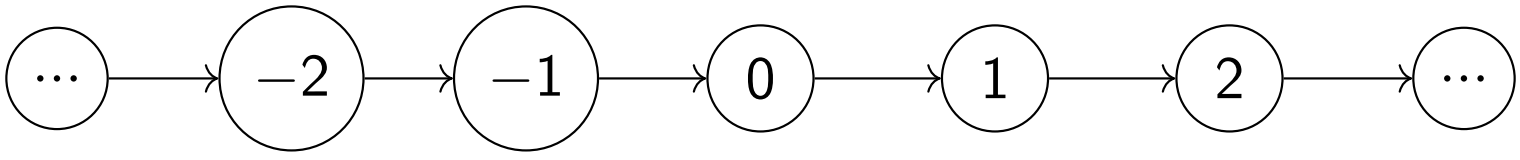
It is clear now that connecting the remaining sources to the sink, or sink the source will result in a strongly connected graph.
As strongly connecting a condensation is equivalent to strongly connecting the original graph, we have strongly connected G.
As at each stage we reduced the number of sources and sinks by 1, we must have used

$$\max(\text{SINK}(C), \text{SOURCE}(C)),$$

edges. $\square$

Infinite Counterexample

The result is not true in the infinite case!



Imperial College London

Algorithm Overview

Steps:

1. Generate the condensation graph by collapsing strongly connected components into single vertices.
2. Iteratively connect sources and sinks to reduce their numbers while preserving weak connectivity.
3. Add edges to connect the remaining sources and sinks, ensuring strong connectivity in the graph.

Theoretical Optimal Complexity:

- $O(V + E)$, where V is the number of vertices and E is the number of edges in the graph.
- Since we need to check every vertex and edge to even determine if the graph is strongly connected

Imperial College London

Depth First Search

Overview:

- DFS starts at a given node and explores each path deeply into the graph until no further nodes can be visited, then backtracks.
- An empty set is initialized to track visited nodes.
- Recursively visit all reachable neighbors from the starting node, marking nodes as visited.
- Once complete, the set contains all successor nodes reachable from the initial source.
- O(V+E) every time we do it, as each vertex and edge is visited once.

Imperial College London

Identifying Strongly Connected Components Using Successors

Process Overview:

1. Perform Depth First Search (DFS) of successors for each node to construct a reachability mapping.
2. Prepare:
   - A container to store SCCs.
   - A set to track processed nodes and avoid redundancy.
3. Traverse each node and its successors:
   - Check for mutual reachability between nodes.
   - Group nodes with mutual reachability into the same SCC.
4. Isolated nodes without mutual reachability are treated as individual components.

Therefore O(V(V+E)) since we require a DFS for each node.

Actual Method Used to Identify Strongly Connected Components

Tarjan's Strongly Connected Component Algorithm:

- Uses a single depth first search to identify strongly connected components.
- Implemented in Networkx condensation package which we use in our fast implementation
- O(V+E)

Imperial College London

Identifying Visions using Depth First Search of Sources

Process Overview:

1. Perform Depth First Search (DFS) of successors for each source, retaining the successor if the vertex is marked as a sink.
2. Prepare:
   - A container to store visions.
3. Traverse each source and its successors:
   - If the successor is marked as a sink, add to the vision for the source.

Therefore O(S(V+E)) since we require a DFS for each source.

Imperial College London

Identifying Visions using Back Propagation

Process Overview:

- Perform Back Propagation (BP) of successors for each source, back propagating the sinks visible to each vertex.
- Prepare:
  - Mark each vertex to record the number of back propagations to it, and the visible sinks.
- From each sink, mark each predecessor with the visible sinks of the current vertex.

Imperial College London

Identifying Visions using Back Propagations Cont'd

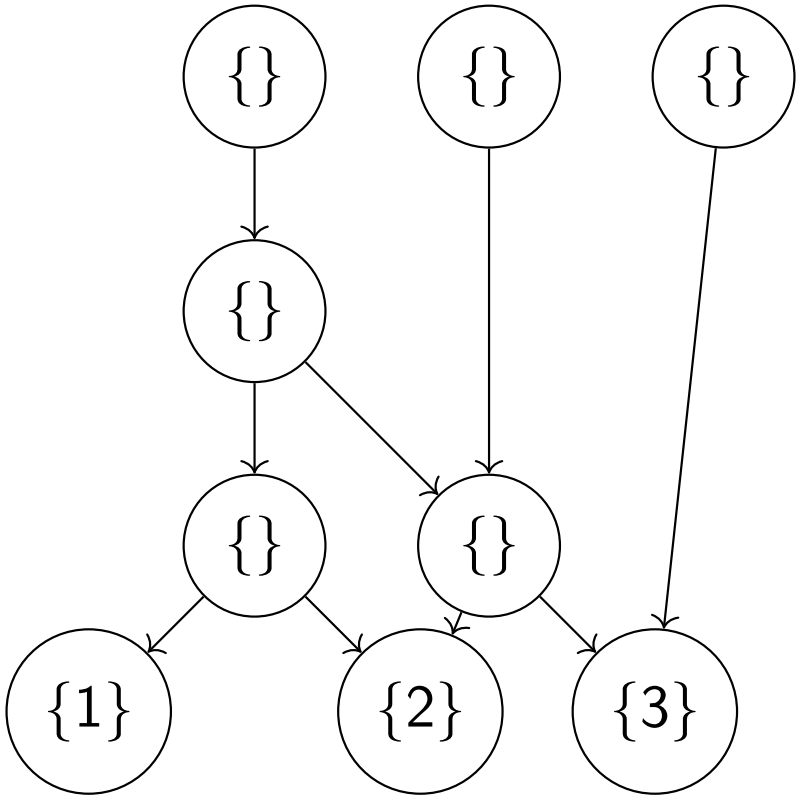
Process Overview Cont'd:

- Once each edge backwards has been reversed. Remove it from the list of sinks to check.
- When an item has the same number of back propagations as its out degree, treat it as a sink. (Since all of the sinks to it will have been removed)
    - Record the visions labelled on each source at the end of the algorithm.

Therefore O(V+E) since we visit each vertex and then traverse backwards along each edge once.

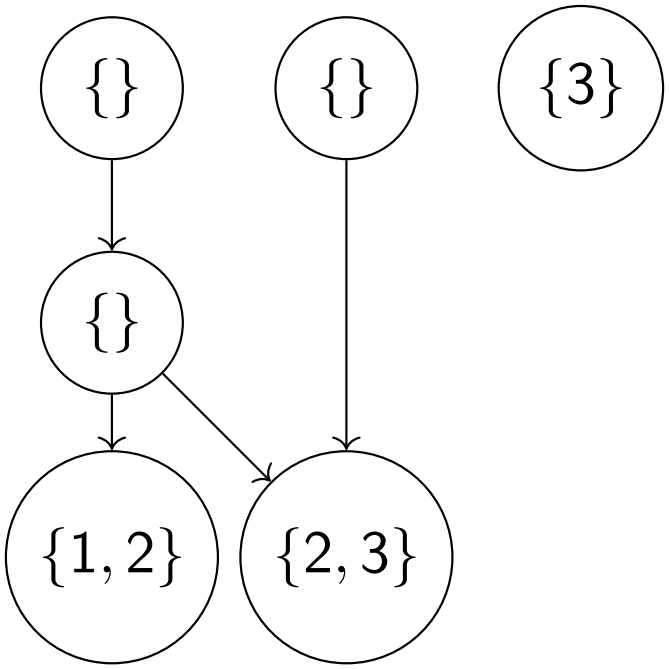
Imperial College
London

Example of Finding Visions using BP 1



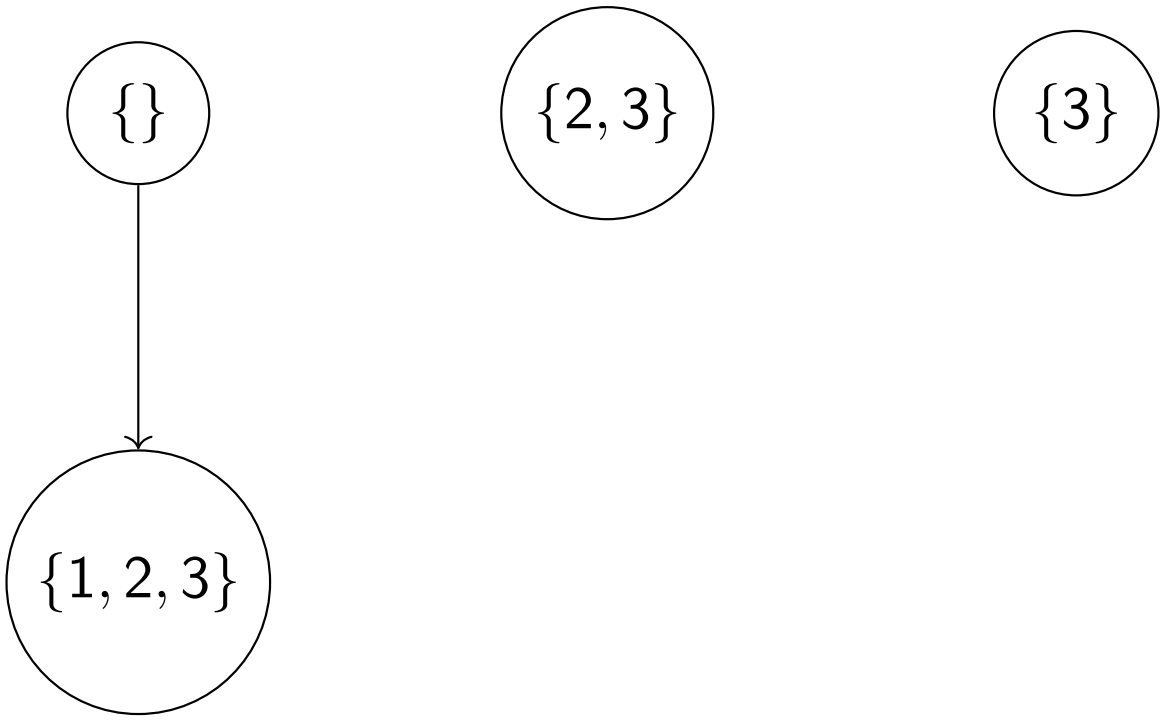
Imperial College London

Example of Finding Visions using BP 2



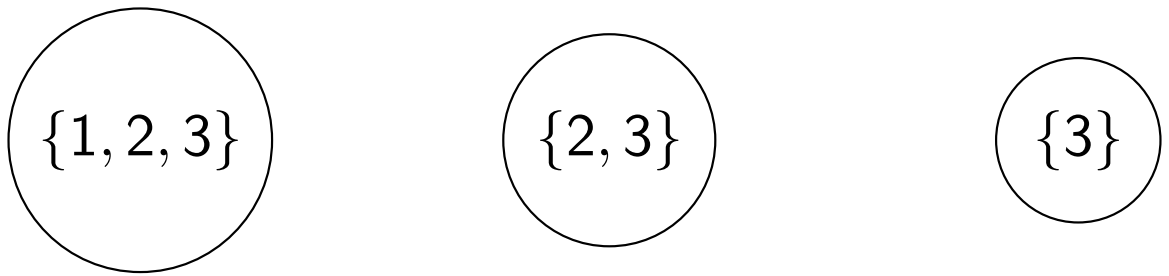
Imperial College London

Example of Finding Visions using BP 3



Imperial College London

Example of Finding Visions using BP 4



Step 1 of the Algorithm

Process Overview:

- Identify sequence of sources to add, such that the union of their visions is strictly increasing in size.
- Prepare:
  - Complete a back propagation search to obtain the sources that can see each sink. (By reversing the edges of the visions algorithm)
  - A list of sinks, marking which is visible and not.
  - A list of sources that have the desired property
  - A list of sinks used to find the sources
- Iterate along the list of sinks.
- If a sink is not visible, find a source that can see it and update the entries of visible sinks. Add the source to the list, and sink to the sink list.

Step 1 of the Algorithm Cont'd

Process Overview Cont'd:

1. As all sinks are visible to sources, this will result in a list of sources with the desired property.
2. Output the list of sources, and the list of sinks to use in the algorithm.

Complexity of Step 1

Consideration of Complexity:

1. Since we have to update the list of visible sinks for every sink in the vision of the source, that step could be as bad as if we add back the same sinks we already knew were visible $O((1 + 2 + ... + H)) = O(H^2)$.
2. As we required two breadth first searches on the condensation, which has at most V vertices and E edges, it was potentially $O(V + E)$. Therefore this step is potentially $O(V + E + H^2)$ but is likely better in practice.
3. In the case of single vertex pointing to sinks, then $H = V - 1$, giving the algorithm a theoretical worst complexity of $O(V^2 + E)$

Overall Complexity Conclusion

- The theoretical best possible complexity to achieve is $O(V + E)$
- We believe we achieve complexity of $O(V + E + H^2)$
- H can be proportional to V, and so if the number of sources of the condensation of the graph is not known, the algorithm may be $O(V^2 + E)$

Imperial College London

Complexity Analysis

We run the algorithm on a series of graphs (networks), and produce figures of the time the algorithm took to complete vs the number of edges, vertices, sources and sinks.

Imperial College London

Complexity Analysis on Synthetic Graphs

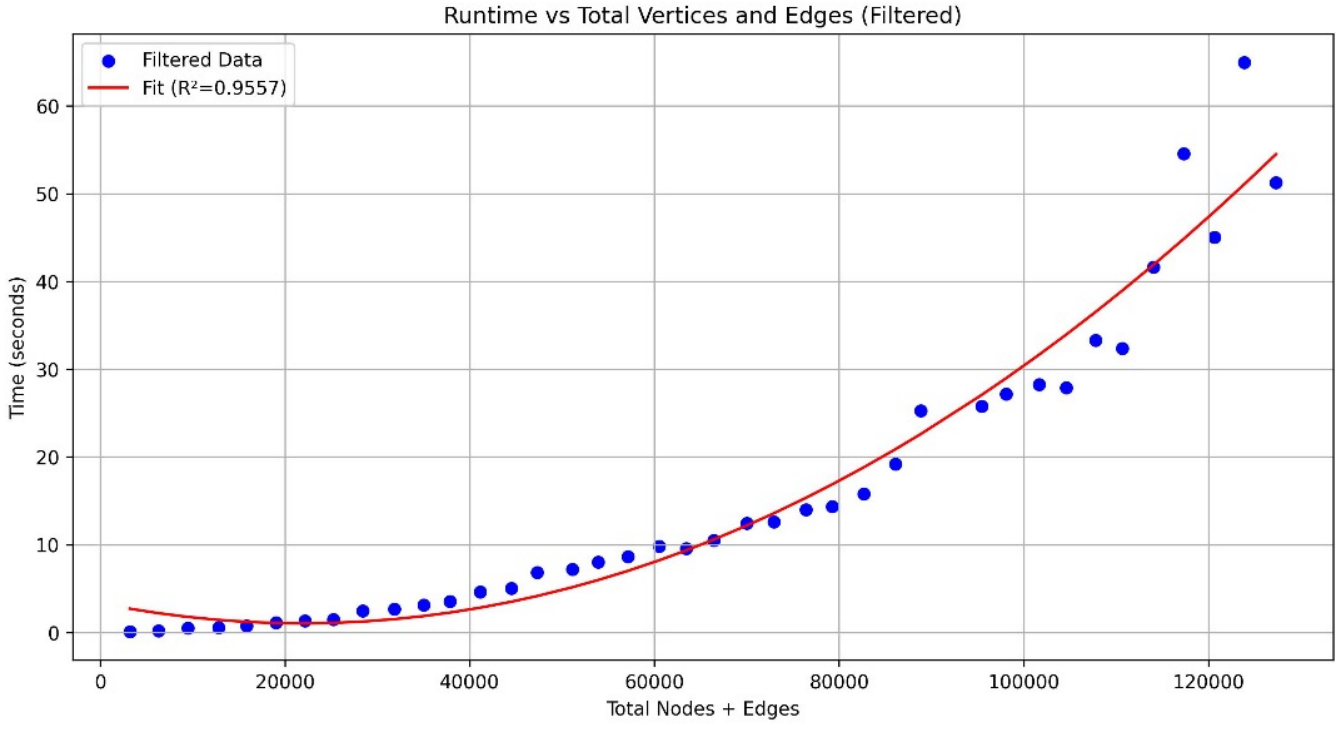


Figure 1: Runtime on Scaled Graphs against total Vertices + Edges with Square Fit Regression Line

Imperial College London

Complexity Analysis on Synthetic Graphs

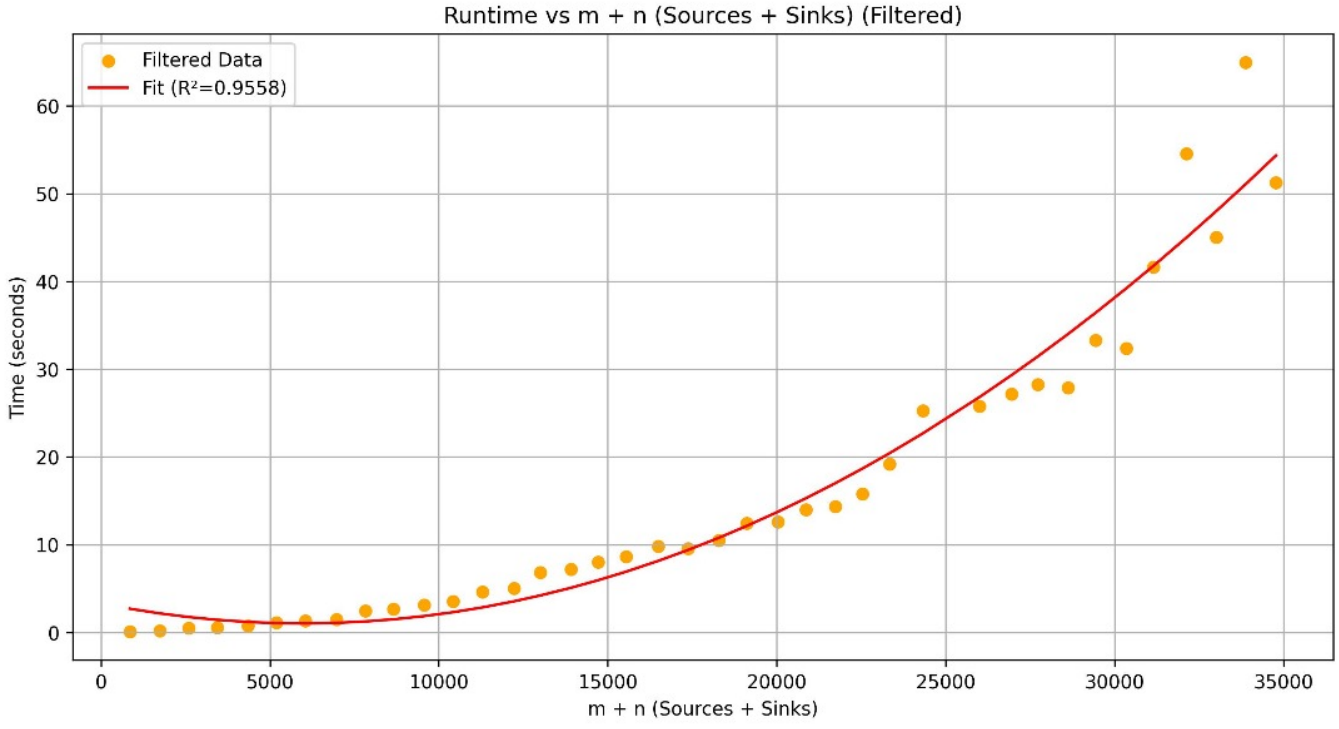


Figure 2: Runtime on Scaled Graphs against total Sources + Sinks with Square Fit Regression Line

Imperial College London

Complexity Analysis on Real World Data

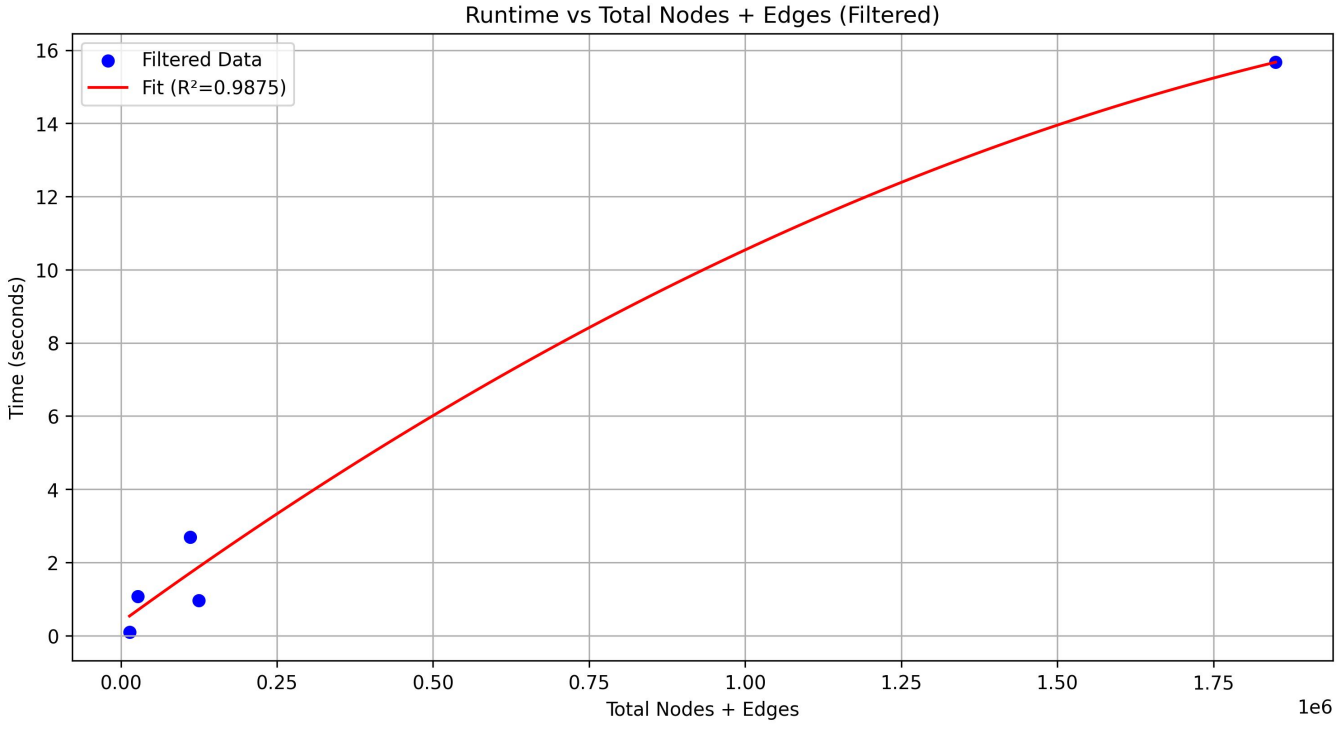


Figure 3: Runtime on a selection of Real World Data Graphs against total Vertices + Edges with Square Fit Regression Line

Imperial College London

Complexity Analysis on Real World Data

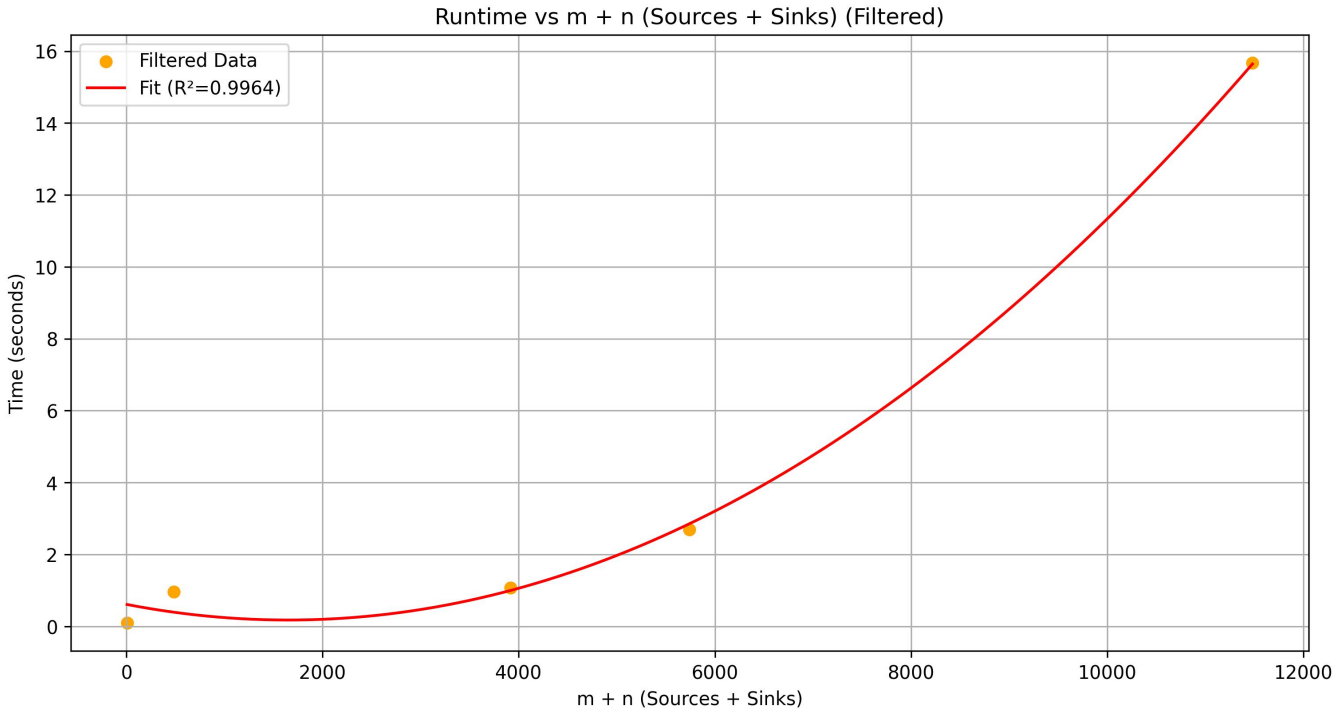


Figure 4: Runtime on a selection of Real World Data Graphs against total Sources + Sinks with Square Fit Regression Line

Data Source

Stanford Large Network Dataset Collection (SNAP):

- SNAP provides a variety of real-world network datasets, including directed networks.
- Examples include datasets from social networks, such as *Social Circles from Twitter*.
- Data is stored in formats such as `.txt` as edge lists, which can be easily read and processed by the networkx Python package.

Imperial College London

Discussion and Limitations 1

Current Implementation:

Current implementation in python relies on set functions a lot, which are slow.

Testing on Real World Data:

Finding suitable real world graphs to test the algorithm on is difficult, and so we did not test on many.

Imperial College London

Discussion and Limitations 2

Speed Up in Practice:
We achieve quite a reasonably fast algorithm. To improve speed on practice, if achieving the exact minimum is not required, it can be applied to components, and then to the network of now strongly connected components.

Connecting the Graph:
We found a theoretical minimum for strongly connecting the graph, and then proved we could always achieve it!

Conclusion

- We proved a hard result, to show the exact minimum to strongly connect any finite directed graph, and a way of computing it.
- $\max(\text{sink}(C), \text{source}(C))$ edges required.
- We wrote an explicit algorithm to compute the edges required, and implemented it
- The complexity is mostly dependent on the number of sources and sinks in the condensation of the graph.
- Complexity is $O(V + E + H^2)$

Imperial College London

References

All references (in particular for the specific real world data graphs used) will be detailed in the full report.