On a fair Multiflow problem

A Major Qualifying Project (MQP) Report
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements
for the Degree of Bachelor of Science in

Mathematical Sciences

By:

Matthew Milton Finley
AKA: Romaji Milton Amulo

Project Advisors:

Brigitte Servatius

Date: April 2021

On a fair Multiflow problem

Romaji Milton Amulo

May 7, 2021

1 Graphs

Definitions of graphs are primarily from Bondy's 1976 introduction to graphs [1], with some conventions added for ease of use. If no cite is applied, it is either from there or my original work.

1.1 Graph Definitions

Definition 1. *A graph, G, is an ordered triplet, $G = (V(G), E(G), \psi_G)$ where $V(G)$ is a set of vertices or nodes (I will use these terms interchangeably), $E(G)$ is a set of edges, distinct from the nodes, ψ_G is a function from edges to unordered pairs of (not necessarily distinct) nodes, called the Incidence Function of G. If $n \in \psi_G(e)$ then n is incident to e and e is incident to n.*

Note, that two different edges can have the same output from the incidence function. I will also denote unordered pairs as $\langle n_1, n_2 \rangle$, to note they may have the same node twice, unlike a set, but are unordered. A graph can have an empty set of edges, though it must have at least one node. While definitions exist that allow completely empty graphs, these graphs are the exceptions to many rules and are not useful.

Definition 2. *Two nodes, $n_1, n_2 \in V(G)$ are adjacent to each other if there exists $e \in E(G)$ such that $\psi_G(e) = \langle n_1, n_2 \rangle$.*

Example 1. *Let G be given by $G = (\{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\}$, $\{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}\}, \psi_G)$, with ψ_G defined as the following map-*

ping:

$$\begin{array}{rcl}
\psi_G(e_1) & = & \langle n_1, n_1 \rangle \\
\psi_G(e_2) & = & \langle n_1, n_2 \rangle \\
\psi_G(e_3) & = & \langle n_2, n_2 \rangle \\
\psi_G(e_4) & = & \langle n_1, n_6 \rangle \\
\psi_G(e_5) & = & \langle n_1, n_6 \rangle \\
\psi_G(e_6) & = & \langle n_2, n_7 \rangle \\
\psi_G(e_7) & = & \langle n_4, n_5 \rangle \\
\psi_G(e_8) & = & \langle n_4, n_5 \rangle \\
\psi_G(e_9) & = & \langle n_4, n_5 \rangle \\
\psi_G(e_{10}) & = & \langle n_8, n_8 \rangle \\
\psi_G(e_{11}) & = & \langle n_8, n_8 \rangle
\end{array}$$

This graph will be commonly used in this section.

From this, a drawing can be made, by making labeled circles for each node, and curves for each edge. The placement of nodes and edges is arbitrary, since it is only a representation of the graph.

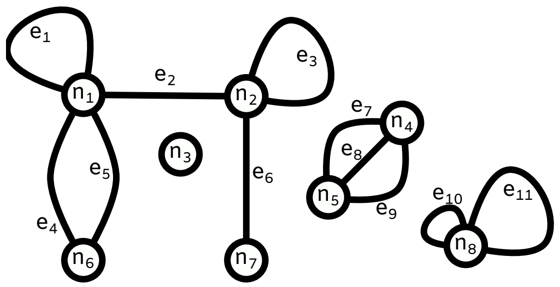


Figure 1: An example drawing of the graph

However, the design of how to draw a graph is not unique and the different ways to draw graphs is an area of research. In this paper, graph drawings will be used simply to aid comprehension and not discussed further.

A graph is finite if both $V(G)$ and $E(G)$ are finite. For ease of notation, if there is only one graph under consideration, the G from the name of each set can be dropped, so $(V(G), E(G), \psi_G\ )$ will be written (V, E, ψ).

Definition 3. *An edge, e, is a loop if $\exists n \in V : \psi(e) = \langle n, n \rangle$.*

Definition 4. *A link is any edge that is not a loop (3). In other words, e is a link if $\exists n_1, n_2 \in V, n_1 \neq n_2 : \psi(e) = \langle n_1, n_2 \rangle$.*

Example 2. *All edges in a graph are links or loops by definition. Loops often need to be treated differently from other edges.*

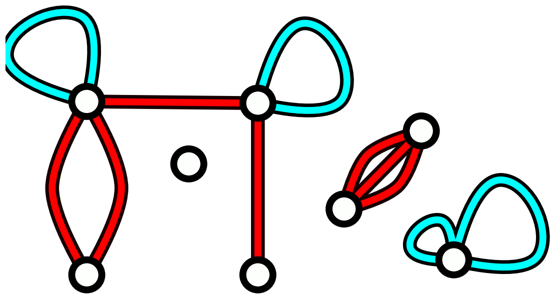


Figure 2: Links are red, Loops are cyan

Definition 5. *Two edges, e_1 and $e_2 \neq e_1$, are a parallel edge pair if $\psi_G(e_1) = \psi_G(e_2)$.*

Definition 6. *A set of edges, $D \subseteq E$ is a parallel edge set if:*

1. *For every edge in D, it is a parallel edge pair (5) with another edge in D.*
2. *There is no edge not in D that makes a parallel edge pair with an edge in D*

Said symbolically, $\forall e_1, e_2 \in D, \psi(e_1) = \psi(e_2)$ and $\forall e_1 \in E \setminus D, \forall e_2 \in D : \psi(e_1) \neq \psi(e_2)$.

A set of parallel edges can also be defined by a pair of adjacent nodes (2), and all edge between them.

Example 3. *Parallel edges are mostly important for Trails (18) since usually they can just be fused together.*

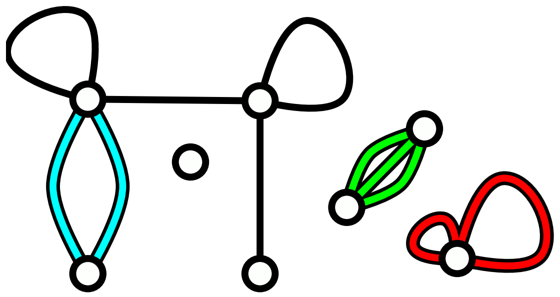


Figure 3: Each set of parallel edges is given a different color. Non-parallel edges are left black.

Definition 7. *A simple graph is a graph (1) with no loops (3) or parallel edges (5).*

Example 4. *Edges can be removed from a graph to make it a simple graph. So, $H = (\{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\}, \{e_2, e_4, e_6, e_8\}, \psi_H)$, with ψ_H defined as the following mapping:*

$$\begin{aligned} e_2 &\to \langle n_1, n_2\rangle \\ e_4 &\to \langle n_1, n_6\rangle \\ e_6 &\to \langle n_2, n_7\rangle \\ e_8 &\to \langle n_4, n_5\rangle \end{aligned}$$

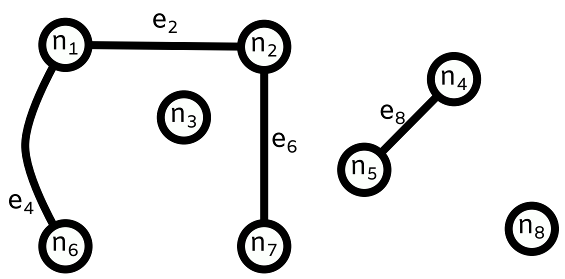


Figure 4: A labeled simple graph.

This is an example of a subgraph (10).

Diestel in his 2017 introduction to graphs [2] uses the definition of an edge as a set of two nodes, because Diestel defines graphs as what I call simple graphs. He instead uses "multi-graph" for those with loops and parallel edges. For clarity, if a property only works for a simple graph, I will label it as one for simple graphs. If I say a property is for all graphs, it is true for both simple and multi-graphs.

Definition 8. *The complete graph with v nodes, K_v, is a simple graph (7) where every pair of nodes is adjacent.*

K_v graph has $\binom{v}{2} = \frac{v(v-1)}{2}$ edges, the maximum for a simple graph with that many nodes.

Example 5. *A common representation of the complete graphs is to draw the nodes as corners of a polygon, with all edges straight.*

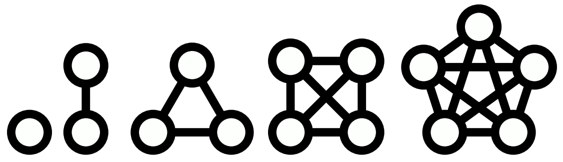


Figure 5: The complete graphs with one to five nodes.

K_4 in particular has three common representations: The one shown above, a tetrahedron, and a variation of the square based one without crossing edges.

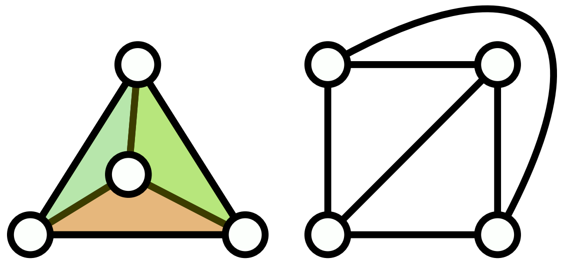
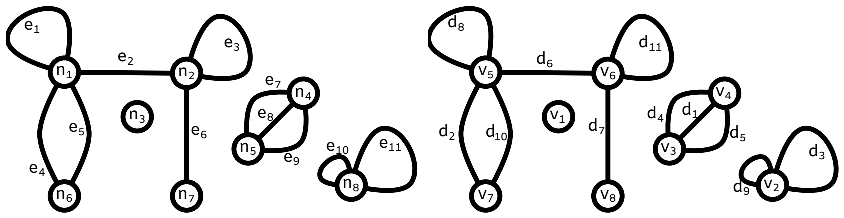


Figure 6: The Tetrahedron and planar graph square drawings of K_4.

Two graphs, G and H, are equal, written as $G = H$ if $V(G) = V(H), E(G) = E(H)$ and $\forall e \in E(G), \psi_G(e) = \psi_H(e)$.

Definition 9. *Two graphs, G and H, are isomorphic, written as $G \cong H$ if $\exists\, \nu : V(G) \to V(H), \varepsilon : E(G) \to E(H)$ that are bijections (every input corresponds to exactly one output, and every output corresponds to exactly one input), such that, $\forall\ e \in E(G), \psi_G(e) = \langle n_1, n_2 \rangle$ then $\psi_H(\varepsilon(e)) = \langle \nu(n_1), \nu(n_2) \rangle$.*

Example 6. *Isomorphism can be seen graphically, as these drawings of graphs G and H are isomorphic at a glance.*



(a) Graph G (b) Graph H

Figure 7: These drawings make it clear that G and H are isomorphic.

The bijective functions in this case are:

$$\nu\left(n_k\right) = \begin{cases} n_1 : & v_5 \\ n_2 : & v_6 \\ n_3 : & v_1 \\ n_4 : & v_4 \\ n_5 : & v_3 \\ n_6 : & v_7 \\ n_7 : & v_8 \\ n_8 : & v_2 \end{cases}, \ \varepsilon\left(e_k\right) = \begin{cases} e_1 : & d_8 \\ e_2 : & d_6 \\ e_3 : & d_{11} \\ e_4 : & d_2 \\ e_5 : & d_{10} \\ e_6 : & d_7 \\ e_7 : & d_4 \\ e_8 : & d_1 \\ e_9 : & d_5 \\ e_{10} : & d_9 \\ e_{11} : & d_3 \end{cases}$$

However, graphs can be drawn in a way that makes them look like they're not isomorphic when they are, and similar looking graphs can be not isomorphic. In this paper though, the math does not rely on which isomorphism of a graph is considered. Graphs will be drawn consistently to make comparisons easier.

Definition 10. *A subgraph H of G, written $H \subseteq G$, is a graph (1) where $V(H) \subseteq V(G), E(H) \subseteq E(G)$ and $\forall\ e \in E(H), \psi_H(e) = \psi_G(e)$*

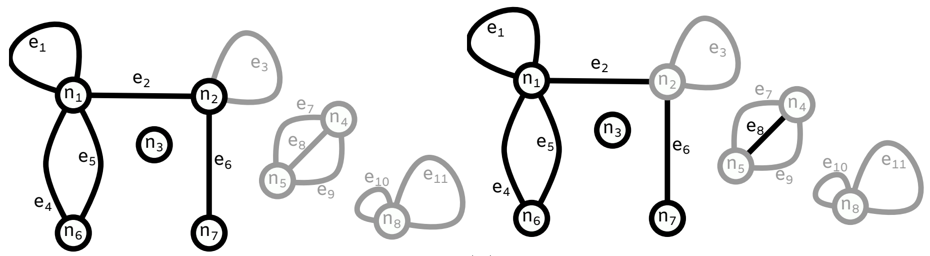
This means that the edges in H can not join nodes not joined in G. ψ_H is only defined on $E(H)$, and can only output nodes found in $V(H)$ Note that every simple graph with v nodes is a subgraph of K_v.

Definition 11. *If $H \subseteq G$ (10) and $H \neq G$, then H is a proper subgraph of G, which is written $H \subset G$. Equivalently, G is a supergraph of H, and if H is a proper subgraph of G, then G is a proper supergraph of H.*

The notation is the same as it is for sets.

If H is a subgraph of G, H must be a graph (1) itself. This means that all edges in H must be adjacent (2) to only nodes in H, and have the same adjacency as they do in G.

Example 7. *If H does not fulfill the shared adjacency condition, then it is not a subgraph.*



(a) A subgraph of G

(b) Not all edges follow the adjacncey condition.

Figure 8: A subgraph and not a subgraph of G. Grey edges and nodes are ones not in H but are in G.

Note that edges 2, 6, and 8 in subfigure b have one or both connections to nodes not in H. This is not allowed in a subgraph.

If ψ_H does not match ψ_G for all edges in H, then H is not a subgraph.

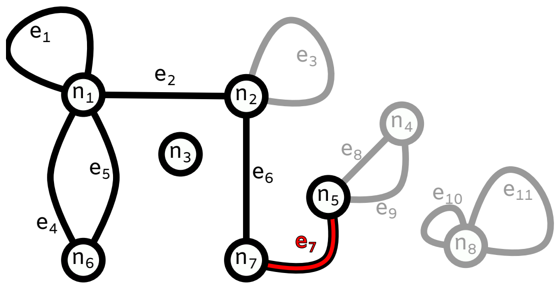


Figure 9: Not a subgraph because of e_7, the highlighted edge, doesn't connect the same nodes as in G.

While the label of e_7 is in both G and H, the label refers to effectively different edges entirely. Similarly, if the edges are relabeled, it is not a subgraph, though this mostly is a technicality, given most graphs in this paper have edges unlabeled.

moved the kinds of subgraph to overflow, as I don't think I need them.

Lemma 1. *If $J \subseteq H \subseteq G$ where J, H, and G are all graphs, then $J \subseteq G$. (10)*

Proof. By definition, $V(R) \subseteq V(H) \subseteq V(G) \implies V(R) \subseteq V(G)$ and $E(R) \subseteq E(H) \subseteq E(G) \implies E(R) \subseteq E(G)$.

By definition of a subgraph, $\forall e \in E(R), \psi_R(e) = \psi_H(e)$ and $\forall e \in E(H), \psi_H(e) = \psi_G(e)$.

Because $E(R) \subseteq E(H)$, $\forall e \in E(R), \psi_R(e) = \psi_H(e) = \psi_G(e)$ □

Given this, the subgraph of a subgraph is a subgraph of the original.

1.2 Weighted Graphs

Definition 12. *A weighted graph, G, is an ordered quartet, $G = (V(G), E(G), \psi_G, w_G)$, which are the vertices, edges, incidence function (1), and the weight function, $w_G : E(G) \to \mathbb{R}$.*

Note that the weight of an edge is another property of the edge itself, and in a subgraph (10), each edge must stay the same weight as it was in the original. Otherwise, all graph properties apply on the "underlying" graph, which is simply the graph without the weight function.

Example 8. *A weighted graph is drawn very similarly to any other graph, but each edge is labeled with the weight of that edge.*

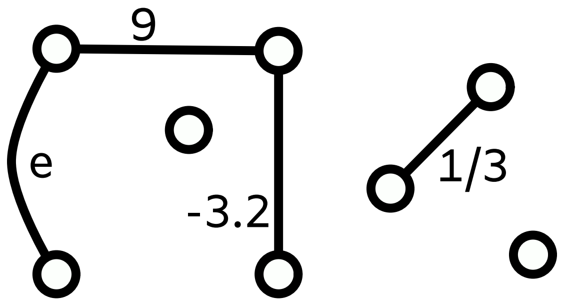


Figure 10: Note that a weight can be any number, even negative or irrational.

1.3 Walks, Trails, Paths, Cycles, Connection

I am citing Professor Ruj's 2014 notes for my definitions related to walks [3].

For this subsection, it is better to use a fully connected (as later explained at 21) graph,

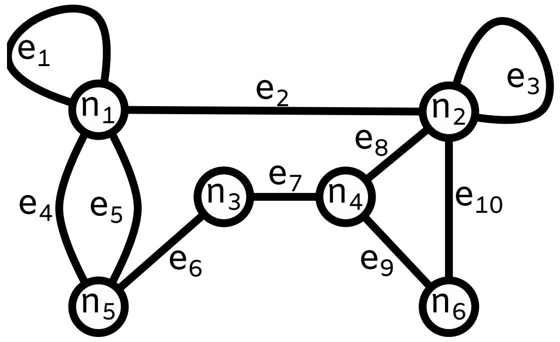


Figure 11: The common graph for this section, labeled

Definition 13. *A walk, W, is an alternating sequence of edges and nodes, $W = v_0, e_1, v_1, \dots e_k, v_k$, where $\forall i \in \{1 \dots k\}, \psi(e_i) = \langle v_{i-1}, v_i \rangle$. A walk is from v_0 (the origin or initial node) to v_k (The terminus or terminal vertex). The length of the walk is k.*

Example 9. *$v_1, e_1, v_1, e_2, v_2, e_8, v_4, e_8, v_2, e_{10}, v_6$ is a walk on the graph above (Fig 11), that looks like*

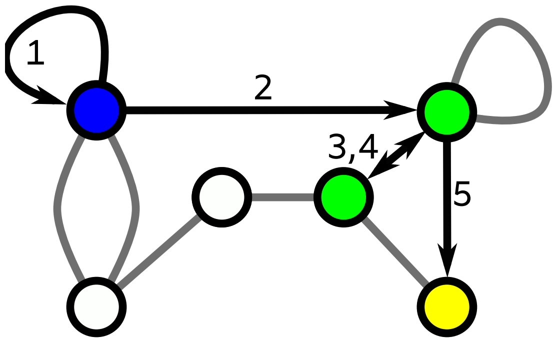


Figure 12: Blue is start, yellow is end, intermediate nodes are green, edges not used are gray, and each edge is labeled sequentially.

$v_1, e_7, v_4, e_8, v_2, e_2, v_1$ is not a walk because e_7 is not incident to v_1

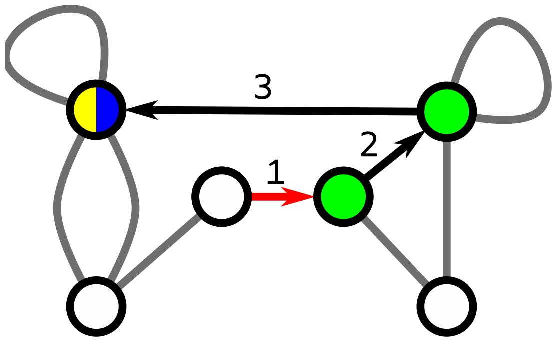


Figure 13: Same conventions, with the highlighted red edge being the error.

Definition 14. *A walk (13), W, has a reverse, W^{-1}, from v_k to v_0, with all the nodes and edges in the reverse order. Formally, $W^{-1} = v_k, e_k, \ldots e_1, v_0$.*

Example 10. *$v_6, e_{10}, v_2, e_8, v_4, e_8, v_2, e_2, v_1, e_1, v_1$ is the reverse of the walk in the previous example (Ex 9)*

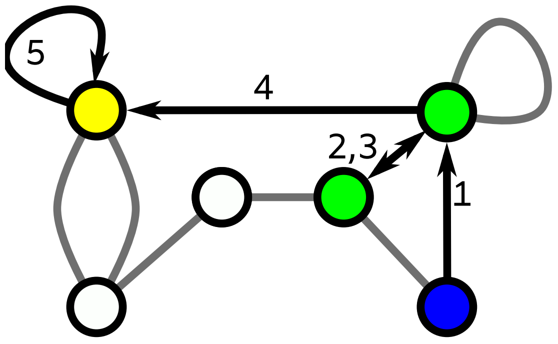
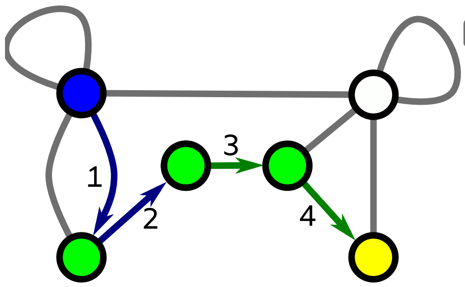


Figure 14: Compare with walk above.

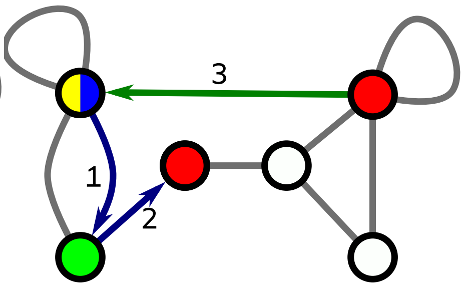
Definition 15. *For walks (13), $W = v_0^W, e_1^W, v_1^W, \ldots e_k^W, v_k^W$ and $A = v_0^A, e_1^A, v_1^A, \ldots e_c^A, v_c^A$, if $v_k^W = v_0^A$, then the concatenation of the walks, $W^\frown A = v_0^W, e_1^W, v_1^W, \ldots e_k^W, v_0^A, e_1^A, v_1^A, \ldots e_c^A, v_c^A$.*

Concatenation is written many ways, I decided on this one to make the difference between a long name and concatenation clear.

Example 11. *Let walks $W = v_1, e_5, v_5, e_6, v_3$, $A = v_3, e_7, v_4, e_9, v_6$, and $L = v_2, e_1, v_1$ be on Fig 11. Then, $W^\frown A = v_1, e_5, v_5, e_6, v_3, e_7, v_4, e_9, v_6$, but $W^\frown L$ is not valid since the origin of L is not the terminus of W.*



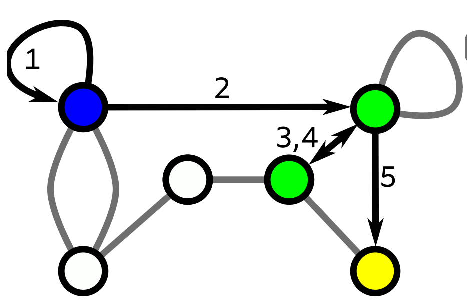
(a) Two walks being joined together (blue edges for W, green for A)



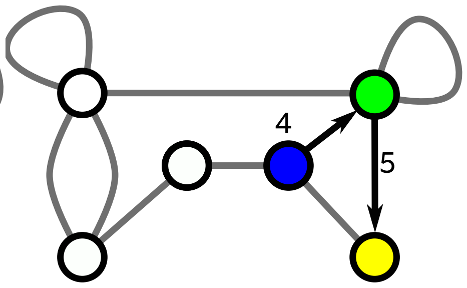
(b) Two walks that don't join together, with the mismatched ends being red. (blue edges for W, green for L)

Definition 16. *A section of a walk (13), $v_i, e_{i+1}, v_{i+1}, \ldots e_j, v_j$, is the (i,j)-section of the walk W, or $W[i,j]$.*

Example 12. *$W = v_1, e_1, v_1, e_2, v_2, e_8, v_4, e_8, v_2, e_{10}, v_6$ is a walk on the standard graph (Fig 11), with the section $W[3,5]$ being $v_4, e_8, v_2, e_{10}, v_6$*



(a) The base walk



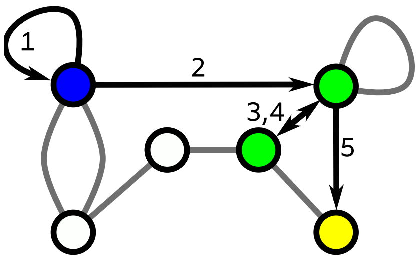
(b) The section is labeled with the numbers of the edges in the original walk

My source uses (v_i, v_j)-section instead of (i, j)-section, however if $\exists v_a = v_i, v_b = v_j, a \neq i, b \neq j, a < b$, then defining the section by node rather than index is ambiguous.

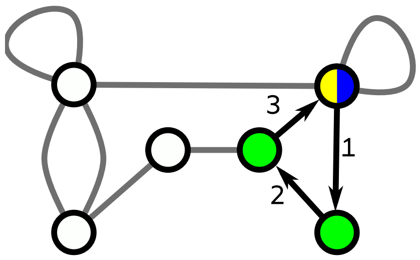
Definition 17. *A walk (13) is closed if $v_0 = v_k$, or the initial and terminal vertex are the same.*

A walk that is not closed is open. All ot the examples I shown before were open.

Example 13. *$W = v_1, e_1, v_1, e_2, v_2, e_8, v_4, e_8, v_2, e_{10}, v_6$ is an open walk on the standard graph (Fig 11), and $A = v_2, e_{10}, v_6, e_9, v_4, e_8, v_2$ is a closed walk on the standard graph.*



(a) The open walk



(b) The closed walk

In a simple graph, the sequence of nodes uniquely define the walk, so it can be defined by this sequence only. However, in a multi-graph this is not always the case. In the case of a multi-graph, a list of nodes where $\forall i \in \{1 \dots k\} \exists e : \psi(e) = \langle v_{i-1}, v_i \rangle$, then the list defines a family of walks, only differing by the edges used.

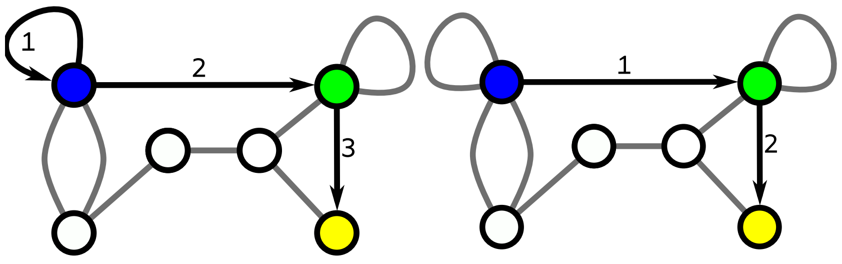
Definition 18. *If no edge is used twice in a walk (13), it is a trail. In a trail, the size of the edge set is the length of the walk.*

Note that in a multi-graph, some families of walks will have some but not all walks in them be trails. This is because going between two nodes twice can use different edges or the same one.

Definition 19. *A path is a trail (18) where every node appears at most once.*

A path can also refer to a subgraph made by only including the edges and nodes used in a path [1], I will call these "path subgraphs" for clarity. Unlike a trail, either all of the walks in a family are paths, or none of them are.

Example 14. *$T = v_1, e_1, v_1, e_2, v_2, e_{10}, v_6$ is a trail, but not a path, and $P = v_1, e_2, v_2, e_{10}, v_6$ is both on the standard graph (Fig 11).*



(a) A trail, note how all edges are used once, even though the initial node is visited twice

(b) A path between the same nodes. The loop is removed so the initial node is visited only once

Definition 20. *A cycle is a closed (17) trail (18) where all nodes except the initial and terminal nodes appear at most once.*

If a walk, W exists between two distinct nodes, s, t, then a path exists between them. Here is a simple algorithm that will convert a walk from s to t, $W_0 = v_{0,0}, e_{1,0}, v_{1,0}, \dots e_{k_0,0}, v_{k_0,0}$, into a path in finite time. First, build the set of pairs of indexes with end nodes, $E_N = \{(i,j) | v_{i,0} = s, v_{j,0} = t\}$. Choose $(i,j) \in E_N : \forall (a,b) \in E_N, |i-j| \leq |a-b|$. If $i < j$, then define

$$W_1 = v_{i,0}, e_{i+1,0}, \dots e_{j,0}, v_{j,0} = v_{0,1}, e_{1,1}, v_{1,1}, \dots e_{j-i,1}, v_{j-i,1}$$

Otherwise, if $i > j$,

$$W_1 = v_{i,0}, e_{i-1,0}, \dots e_{j,0}, v_{j,0} = v_{0,1}, e_{1,1}, v_{1,1}, \dots e_{i-j,1}, v_{i-j,1}$$

(See reverse walks (14) and walk segments (16) for more details.)

Now, take a look at your current walk, W_n, and let k_n be the length of that walk. (On your first time, $n = 1$). If W_n is a path, you're done.

If not, find two indexes, $i < j$, such that $v_{i,n} = v_{j,n}$, where the n is which walk these nodes are on. Then, form W_{n+1} by connecting the walk from $v_{0,n}$ to $v_{i,n}$ with the walk from $v_{j,n}$ to $v_{k_n,n}$. This will reduce the length by $j - i$, and remove at least one closed walk (17) from the walk from s to t. Since the original walk is of finite length, there can only be a finite number of closed subwalks, so the process is finite.

Definition 21. *A graph, G, is connected if $\forall v_1, v_2 \in V(G) \exists P \subseteq G : v_1, v_2 \in V(P)$ and P is a path subgraph [19], or G has one or fewer nodes.*

A graph that is not connected is disconnected.

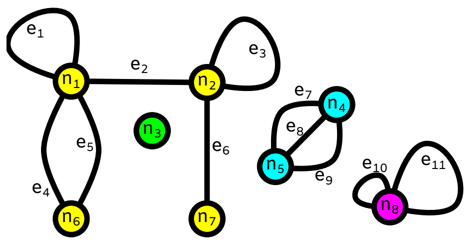
If a graph is disconnected, it is made out of two or more "components".

Definition 22. *A component is subgraph (10) with three properties:*

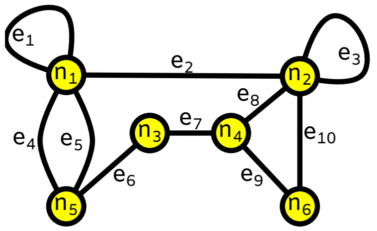
1. *The subgraph is connected (21).*
2. *The subgraph is vertex-induced (**??**).*
3. *There does not exist a node that can be added to the subgraph without disconnecting it (or there's no more nodes to add).*

If a graph is connected, then it is the only component of itself.

Example 15. *The two graphs I've used in this section have different numbers of components. The first graph used (Ex 1) has four components and is disconnected. The second (Fig 11) is connected and has one component.*



(a) This graph has four components, each with a different node color



(b) This graph has only one component, so all nodes are colored yellow

It is often useful to consider how "easy" it is to disconnect a connected graph (21). This can be quantified by the vertex-connectivity (23) (sometimes called just "connectivity") and edge-connectivity (25).

Definition 23. *The vertex-connectivity of a graph G, written $\kappa(G)$, is the minimum number of nodes needed to be removed from a graph to disconnect (21) it.*

A set of $\kappa(G)$ that when removed from the graph disconnects G is a minimal cutset of G

Definition 24. *A graph, G, is k-vertex-connected if $k \leq \kappa(G)$ (23).*

Any graph that has a spanning subgraph (**??**) that is the complete graph (8) has vertex-connectivity $|V(G)| - 1$ by definition. This is because there is no way to disconnect the complete graph by deleting nodes, since a single node is still connected, and vertex-connectivity must be less than the number of nodes. Because vertex-connectivity must be less than the number of nodes, a single node has vertex connectivity zero and is the only connected graph with vertex-connectivity zero.

Removing a node from a graph with positive vertex-connectivity is not guaranteed to reduce the connectivity, as the connectivity is the minimum number needed.

Definition 25. *The edge-connectivity of a graph G, written $\lambda(G)$, is the minimum number of edges needed to be removed from a graph to disconnect (21) it.*

Definition 26. *A graph, G, is k-edge-connected if $k \leq \lambda(G)$ (25).*

A graph with one node is the only connected graph with edge-connectivity zero.

2 Directed Graphs

new cite for this section: [4]

2.1 Digraph Notation

Definition 27. *A directed graph, or a digraph, D, is the same as a graph, except instead of $E(D)$ it has $A(D)$, or a set of arcs. The difference between arcs and edges is that while edges are an **unordered** pair of nodes, an arc, a represents an **ordered** pair, $\psi_D(a) = (v_1, v_2)$, from the tail v_1 (or initial vertex) to the head v_2 (or terminal vertex).*

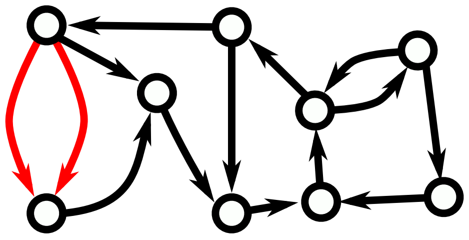
Definition 28. *A pair of arcs, a_1 and $a_2 \neq a_1$, is a parallel arc pair if $\psi_D(a_1) = \psi_D(a_2)$.*

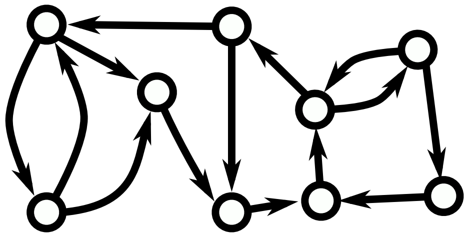
This is similar to a parallel edge pair (5) but for Digraphs. Note that two arcs can go between the same two nodes and not be parallel if they have different directions.

Definition 29. *A digraph is strict if it has no pairs of parallel arcs (28), or loops (3).*

Note that loops are again defined as an arc connecting a node to itself, just as in graphs, a loop is an edge that connects a node to itself.

Example 16. *The first digraph has a parallel arc and the second is strict.*


(a) Red highlights the paralell arcs.


(b) The reversal of one arc makes the digraph strict.

Definition 30. *The complete digraph with v nodes, κ_v, is a strict digraph (29) with every possible arc.*

In other words, $\forall v_1, v_2 \in V(\kappa_v) \exists a \in A(\kappa_v) : \psi_{\kappa_v} = (v_1, v_2)$. This is not the same thing as a "Tournament", which only has one arc per **unordered** pair of nodes, while a complete digraph has one arc per **ordered** pair of nodes. Tournaments are not a structure used in this paper, but do come up in others as a kind of "complete digraph".

Example 17. *The drawings of the complete digraphs shown are meant to resemble the ones for complete graphs (5).*

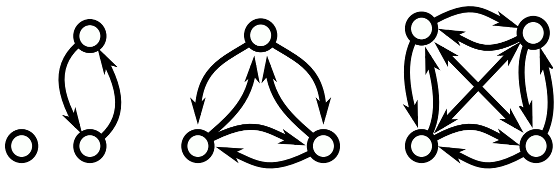


Figure 21: The first 4 complete digraphs. The double arrow lines represent two arcs going in opposite directions.

Definition 31. *A subdigraph, F, of digraph (27) D, written $F \subseteq D$, is a digraph where $V(F) \subseteq V(D), A(F) \subseteq A(D)$ and $\forall a \in A(F), \psi_F(a) = \psi_D(a)$*

This means that the arcs in F must be identical to arcs in D, including ordering, similar to subgraphs (10). In the same way, ψ_F is only defined on $A(F)$ and can only output nodes found in $V(F)$.

The following functions were inspired by definitions from Ms. Fonoberova's 2012 handbook [5] and Mr. Schrijver's 1993 journal article [6], but adapted to the notation of this paper.

Definition 32. *The tail function, $A^- : V \to 2^A$, is*
$A^-(v_1) = \{a | a \in A \wedge \exists v_2 \in V : \psi(a) = (v_1, v_2)\}$*, or the set of all arcs with tail v_1.*

Definition 33. *The head function, $A^+ : V \to 2^A$, is*
$A^+(v_1) = \{a | a \in A \wedge \exists v_2 \in V : \psi(a) = (v_2, v_1)\}$*, or the set of all arcs with head v_1.*

Example 18. *Consider a node in a digraph, and the head and tail functions. The sets of arcs returned can be shown visually with coloring.*

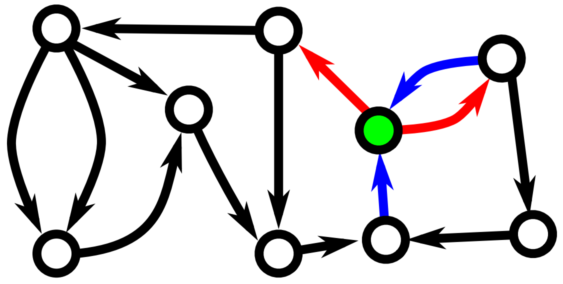


Figure 22: For the green node, the arcs in red have tail at that node and the ones in blue have head at that node.

2.2 Weighted Digraphs

Definition 34. *A weighted digraph, D, is an ordered quartet, $D = (V(D), A(D), \psi_D, w_D)$, where the first three are the same as a digraph (27) and $w_D : A(D) \to \mathbb{R}$.*

Example 19. *A weighted digraph looks the same when drawn as a weighted graph, but with arcs instead of edges. This means that the weight from v_1 to v_2 might be different than v_2 to v_1*

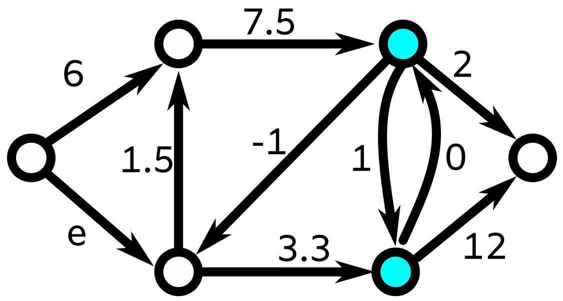


Figure 23: Notice that the weight of the arcs between the two cyan nodes are different.

Definition 35. *The underlying digraph (27) D of a weighted digraph (34) D', is: $D = (V(D'), A(D'), \psi_{D'})$*

The underlying digraph is the digraph without weights.

In this paper, it will be useful to consider transformations on a weighted digraph that change the weights of the arcs but not the structure of the digraph. These are scalar multiplication (or "scaling") and weighted digraph addition (and conversely, subtraction).

Definition 36. *For a weighted digraph (34) D, and $x \in \mathbb{R}$, $x \cdot D = D'$ is a weighted digraph where: $V(D') = V(D), A(D') = A(D)$ and $\forall a \in A(D'), w_{D'}(a) = x \cdot w_D(a)$*

Definition 37. *If weighted digraphs (34) D_1, D_2 have the same underlying digraph (35), then $D_1 + D_2 = D_\Sigma$ is a weighted digraph where: $V(D_\Sigma) = V(D_1), A(D_\Sigma) = A(D_1)$ and $\forall a \in A(D_\Sigma), w_{D_\Sigma}(a) = w_{D_1}(a) + w_{D_2}(a)$*

Example 20. *Consider the following weighted digraph:*

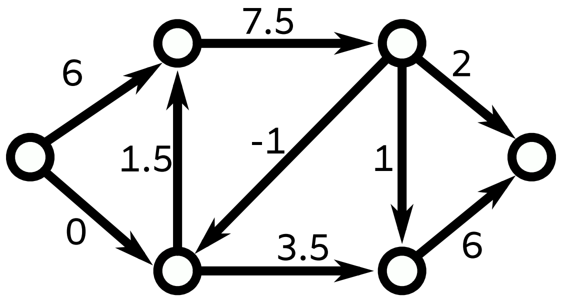


Figure 24: A weighted digraph with positive, negative, and zero entries.

Then, multiply this weighted digraph by 2, which means multiplication of the weight on each arc by two.

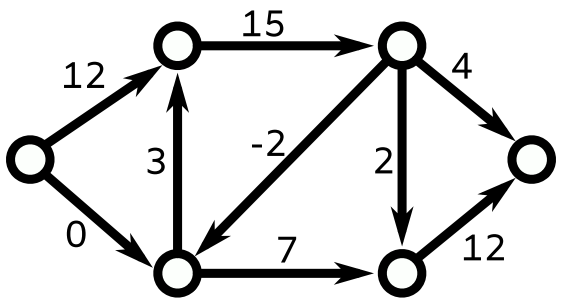


Figure 25: A weighted digraph with positive, negative, and zero entries.

Example 21. *Consider the following weighted digraph, call it A.*

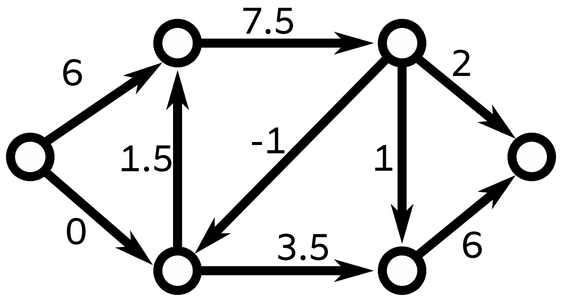
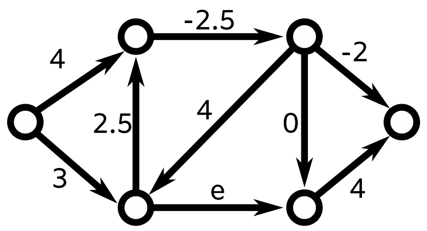
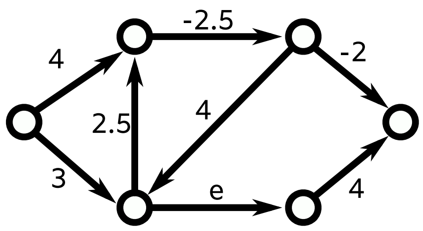


Figure 26: Weighted Digraph A.

Then consider trying to add either of the following digraphs.



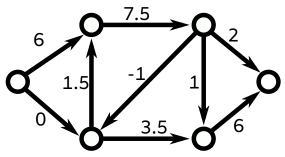
(a) Weighted Digraph X



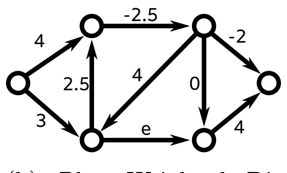
(b) Weighted Digraph Y

Figure 27: Notice that these are the same except for a zero arc.

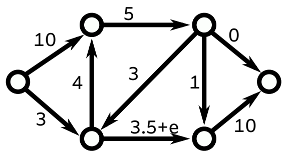
It is important to note that only weighted digraph X can be added to the weighted digraph from the previous example, as the underlying digraphs (35) must be the same, since the addition must occur for all arcs.



(a) Weighted Digraph A



(b) Plus Weighted Digraph X



(c) equals the sum.

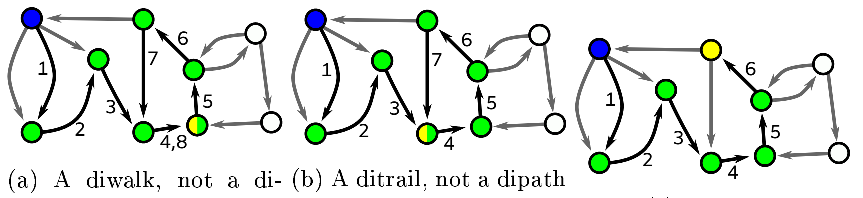
Note that no arcs are added or removed in addition, only the weights of the arcs change. You are not "adding" new arcs, instead adding the weight functions.

2.3 Diwalks, Ditrails, Dipaths, Dicyles, Diconectivity

Definition 38. *A directed walk or diwalk, W, of length k, is an alternating sequence of arcs and nodes, $W = v_0, a_1, v_1, \ldots a_k, v_k$, where $\forall n \in \{1 \ldots k\}, \psi(a_n) = (v_{n-1}, v_n)$.*

A directed trail or ditrail, a directed path or dipath, and a directed cycle or dicycle all have equivalent definitions, but starting from diwalks, rather than walks. I often will refer to dipaths as "paths" when dealing with directed graphs.

Example 22. *For each one, note that the direction walked is always the direction of the arc. This also means the reverse is not defined.*



(a) A diwalk, not a ditrail as arc 4 and 8 are the same

(b) A ditrail, not a dipath as the ending node is visited twice

(c) A dipath

Figure 29: As a reminder, blue is the start of the walk, green are intermediate nodes, and yellow is the end.

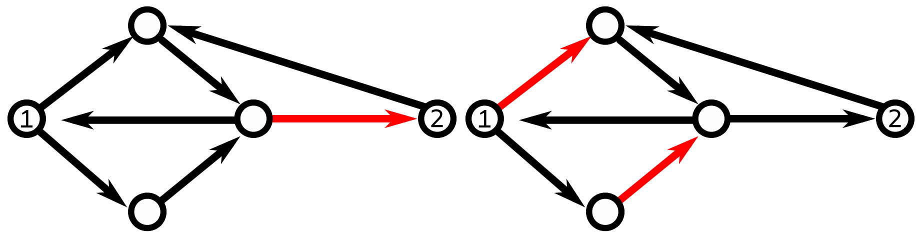
Definition 39. *If there exists a dipath (38) from v_1 to v_2, v_2 is reachable from v_1 in D.*

Definition 40. *For nodes v_1 and v_2 in D (27), a minimal cutset, c, is a set of arcs where:*

1. *Removing all arcs in c from D makes v_2 not reachable from v_1 in D (39)*
2. *Removing all but one arc in c from D will still have v_2 reachable from v_1 in D.*

In other words, a minimal cutset will make one not reachable from the other, but anything less than that will not do the same.

Example 23. *Consider the following digraph with v_1 and v_2 labeled. Here are two minimal cutsets, that make v_2 not reachable from v_1.*



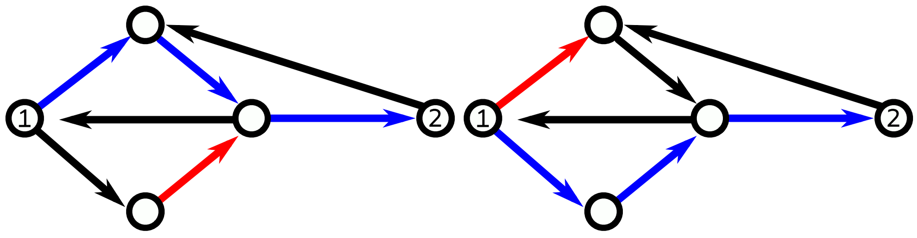
(a) Minimal Cutset 1

(b) Minimal Cutset 2

Figure 30: Red highlights the members of the cutset. Note that while cutset 2 is larger, it is still minimal.

It is easy to see there is no dipath (38) from v_1 to v_2 that does not use an arc in both cutsets. However, there is still a dipath from v_2 to v_1, which is a different question.

Cutset 2 also shows that a minimal cutset does not have to be the smallest cutset, only that removing an arc would make it not a cutset.



(a) One arc removed from set 2 (b) The other arc removed from set 2

Figure 31: Notice, removing either arc would make it not a cutset. Path labeled with blue arcs.

The other way a set can be not a minimal cutset is if it has an arc that is not needed.

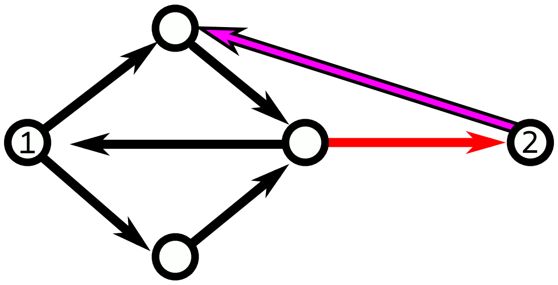


Figure 32: The purple arc can be removed to get minimal cutset 1.

This is still "a cutset", but not a minimal one. Minimal cutsets are important for later, so this distinction is important.

Definition 41. *If v_1 is reachable (39) from v_2 and v_2 is reachable from v_1, then they are diconnected.*

Two nodes are diconnected if there is a closed ditrail that contains both of them.

Dicomponents and diconnected graphs can be defined in the same way as components and connectivity, since diconnectivity is independent of order of nodes, like connectivity (and unlike reachability).

3 Networks

3.1 Networks and Loads

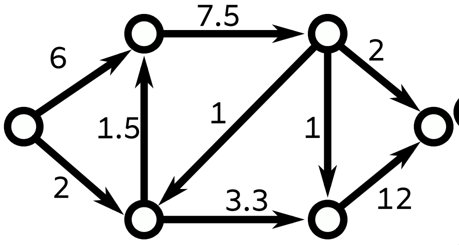
Definition 42. *A Network, N is a weighted directed graph (34), with non-negative rational weights, called capacities.*

Bondy's chapter on networks in his 1976 book [7] says that all capacities must be integers, but all other definitions I've found do not have that restriction.

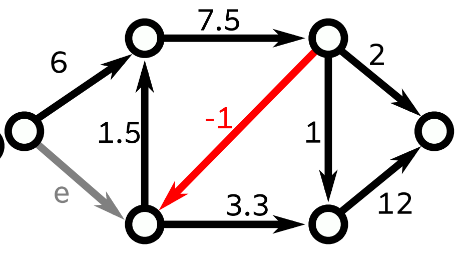
However, there are some cases where this restriction can help, such as converting to a network where all arcs have capacity 1. They also might naturally be integers in discrete applications. In this paper, I will instead assume that all capacities are rational numbers, as computationally, all numbers are rational. Additionally, the network may be scaled (36) to one where all capacities are integers.

For a network, the "capacity" of an arc is the same as its weight, and I will use the terms interchangeably. This means that scaling (36), addition, and subtraction (37) are all defined for networks, with the additional constraint that the final result must also be a network. A network may only be scaled by a non-negative rational number to remain a network. If the number is negative, all weights would become non-positive, which is not allowed for any network with non-zero weights. If the number is irrational, all non-zero entries would become irrational, so this is not allowed if the network has non-zero weights.

Example 24. *A network is a weighted digraph with two constraints: Non-Negative arc weights and rational arc weights.*



(a) A network



(b) The red arc is negative and the grey arc is irrational

Figure 33: A network and not a network.

Definition 43. *A load function, L, on a network N (42), is a function from $A(N)$ to $\mathbb{Q}$ where:*

$$\forall a \in A(N), \begin{cases} if\ w(a) = 0 : L(a) = 0 \\ if\ w(a) > 0 : \ L(a) \geq 0 \end{cases}$$

Example 25. *First, a load function. The capacity will be shown with black numbers, and the load will be blue and in a box.*

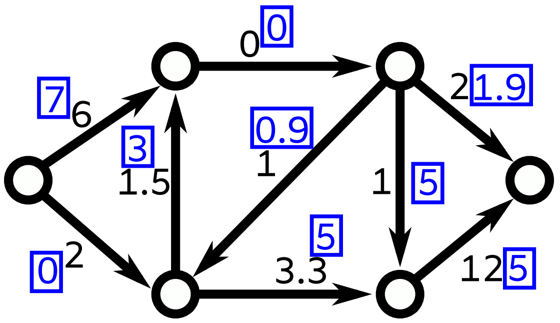
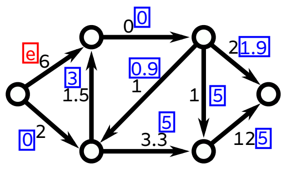
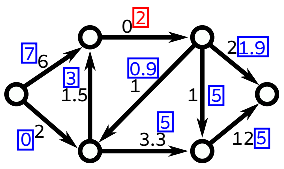


Figure 34: An arbitrary load function. Note how it is rational and non-negative everywhere.

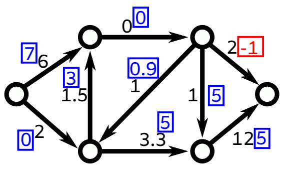
However, not any such way of assigning numbers to arcs relates to a load function. There are three pitfalls that can be run into.



(a) The load function must be rational



(b) The load function must be zero on a zero arc



(c) The load function can't be negative

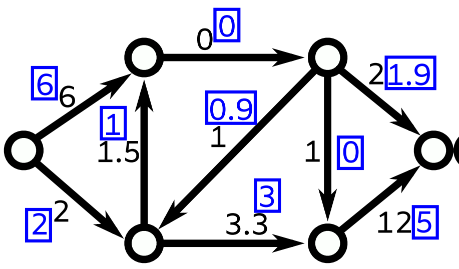
Figure 35: The red number shows the incorrect value

Note that the "zero on a zero arc" is the only condition that depends on the capacity of an arc.

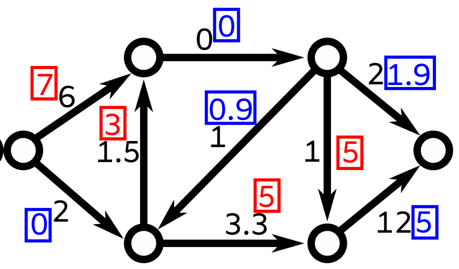
Definition 44. *A load function (43) L on a network N (42) is feasible if* $\forall a \in A, L(a) \leq w(a)$.

The choice of the word "feasible" is a deliberate relation to the word being used in multi-flow contexts, where the condition has a similar meaning. I will use "Feasible on" when there are multiple networks to be concerned about, and just "feasible" if it is clear what network is being considered.

Example 26. *A load being feasible is a way of seeing if it can "fit" into the network or it is "too big".*



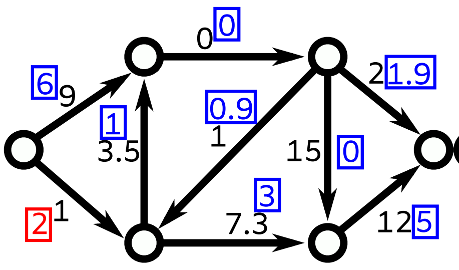
(a) All are less than the capacity



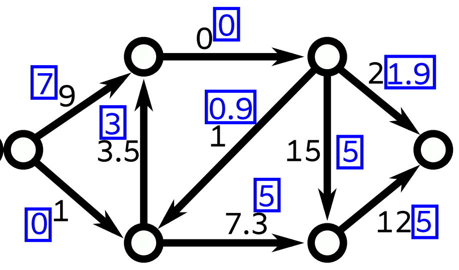
(b) Red numbers are too big

Figure 36: The second one is still a load function, it is just not feasible on this network.

It is important to note that feasibility is an interaction between the network and the load function. If the capacity of the network changes, the feasibility of the load function changes as well for that network.



(a) The red number makes this load too big now



(b) This network is big enough to hold this load

Figure 37: On this changed network, the two loads changed feasibility.

This is an important point to remember, especially when multiple networks with the same underlying digraph exist.

3.2 Utilization and Scaling

A load function, like a network, can be added, subtracted, or scaled, with the same restrictions on scaling and subtraction as a network. Again, proof that these restrictions must exist for the result to be a load function is left to the reader.

Definition 45. *The utilization of a load function(43) L on a network N (42), is the smallest non-negative number, $Ut(L,N)$, where L is feasible (44) on $Ut(L,N)\cdot N$.*

An alternate form of this is very useful in computation, which I will prove below.

Lemma 2. *If N has at least one arc with positive capacity,*

$$Ut(L,N) = \max_{\substack{a \in A(N): \\ w(a) > 0}} \left\{ \frac{L(a)}{w(a)} \right\}$$

Proof. Define $B = \{a \in A(N) | c_N(a) > 0\}$ and $U = Ut(L,N)$. Then, the symbolic form to prove becomes:

$$U = \max_{a \in B} \left\{ \frac{L(a)}{w(a)} \right\}$$

Because we know L is feasible on $U \cdot N$ by definition, begin with the definition of feasibility, restricted to the set B. We can do this restriction since $\forall a \notin B, w(a) = L(a) = 0$ and these are always true for any U. This becomes:

$$\forall a \in B, L(a) \leq U \cdot w(a)$$

Divide both sides by $w(a)$ since by definition, it is never zero in B.

$$\forall a \in B, \frac{L(a)}{w(a)} \leq U$$

Since U is by definition the smallest such number with this property, it must be the maximum of the left hand side, which is exactly the symbolic form. □

There are numerous useful properties of this function in relation to feasibility, scaling, and addition. These are all presented without proof (at the moment). Let N be a network (with at least one positive arc), N' be an alternate weighting of that network, $\mathbb{Q}^+ = \{x \in \mathbb{Q} | x > 0\}$, $\mathbb{L}$ be all load functions on N, and $F(L)$ is 1 if L is feasible on N and 0 if it is not.

1. Triangle Inequality

$$\forall L_1, L_2 \in \mathbb{L},\ Ut(L_1, N) + Ut(L_2, N) \geq Ut(L_1 + L_2, N) \geq Ut(L_1, N)$$

2. Feasibility-Utilization Relation

$$\forall L_1, L_2 \in \mathbb{L} : F(L_1) = 1 \wedge F(F_2) = 0,\ Ut(L_1, N) \leq 1 < Ut(L_2, N)$$

3. Scalar Multiplication (load)

$$\forall L \in \mathbb{L} \forall x \in \mathbb{Q}^+, Ut(x \cdot L, N) = x \cdot Ut(L, N)$$

4. Zero Load Property

$$\forall L \in \mathbb{L}, Ut(0 \cdot L, N) = 0$$

5. Scalar Multiplication (network)

$$\forall L \in \mathbb{L} \forall x \in \mathbb{Q}^+, Ut(L, x \cdot N) = \frac{Ut(L, N)}{x}$$

6. Expanded Network Inequality

$$\forall L \in \mathbb{L}, Ut(L, N + N') \leq Ut(L, N)$$

All of these properties are important to keep filed away, as they're important for working with this function.

3.3 Flows, Normal and Unrestricted

Definition 46. *On a given network (42), N, a flow (or Restricted Flow), a flow F is an ordered triple,* $(L_F, s(F), t(F))$*, where* $L_F(a)$ *is the flow's load function,* $s(F)$ *is the source node, and* $t(F)$ *is the terminal (or sink) node.*

The load function of a flow is subject to two additional conditions:

1.
$$\forall n \in V(N) \setminus \{s(F), t(F)\} \rightarrow \sum_{a \in A^-(n)} L_F\left(a\right) - \sum_{a \in A^+(n)} L_F\left(a\right) = 0$$

2.
$$\forall a \in A(N) \rightarrow 0 \leq L_F(a) \leq w(a)$$

These are called the conservation constraint and the capacity condition.

In some sources, a flow is restricted to integer values, which is useful if the flow comes in discrete units. In this paper, a flow is a rational valued function.

Some definitions allow for multiple sources and sinks, but I go with [8] which defines only one source and sink. Multiple sources can be converted into a single source by adding a "supersource" [8], with an arc from the "supersource" to each existing source with the capacity of the sum of the capacities of all arcs leaving that source. Equivalently, multiple sinks can be converted to a single sink by adding a "supersink", with an arc to the "supersink" from each existing sink, with the capacity of the sum of the capacities of all arcs entering that sink.

A flow F has a **value**, notated by $\langle F \rangle$, which is the amount of flow entering the network at the source, symbolically

$$\langle F \rangle = \sum_{a \in A^-(s(F))} L_F\left(a\right) - \sum_{a \in A^+(s(F))} L_F\left(a\right)$$

Bondy [7] uses *val F* and Schrijver [6] uses $value(F)$, but I decided to use my own, shorter notation, since the value of a flow will come up often. The value of a flow can be zero or negative, but assume positive value flows unless otherwise stated.

Flows are a standard definition in network problems, however, in this paper it's useful to think of them as a special case of a new object, the unrestricted flow.

Definition 47. *An Unrestricted Flow or "uflow", U, on network N (42) is an ordered triple, $U = (L_U(a), s(U), t(U))$, where $L_U(a)$ is the uflow's load function, $s(U)$ is the source node, and $t(U)$ is the terminal node.*

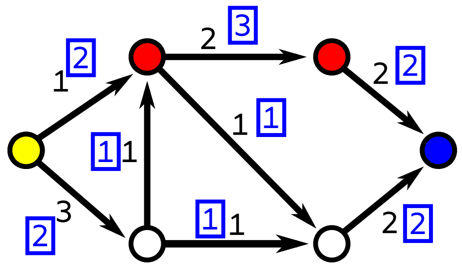
The uflow load function only needs to satisfy the conservation constraint, which is:

$$\forall n \in V(N) \setminus \{s(U), t(U)\} \rightarrow \sum_{a \in A^-(n)} L_U(a) - \sum_{a \in A^+(n)} L_U(a) = 0$$

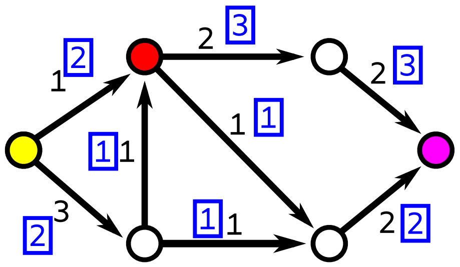
All flows are also uflows, because a uflow's load function is less restricted than a flow's load function.

Example 27. *The core of a flow or uflow is its load function. The Conservation Constraint can be broken in two kinds of ways, which I call "internal" and "external". An internal break of the Conservation Constraint is one where the amount of flow in and out of the network is still the same. An external break, by contrast, makes the amount of flow in not equal to the amount of flow out.*

The two diagrams below will show an internal and an external break of the conservation constraint, with the violating nodes being marked with red. Like with paths, the source will be blue and the sink will be yellow, and the load function will be numbers inside rectangles.



(a) Note the error begins and ends inside



(b) Note that there's only one error node, but the out is not the same as the in (shown with a purple end node)

Figure 38: Both of these violate the Conservation Constraint at least once.

The Conservation Constraint is a very strict condition. It also depends entirely on the source and sink, as those are the two nodes where the Conservation Constraint must apply.

A flow is subject to one more constraint than a uflow, which is that a flow must be less than or equal to the capacity on all arcs. This is shown below with the purple load number being the only one bigger than the capacity, hence breaking the Capacity Condition.

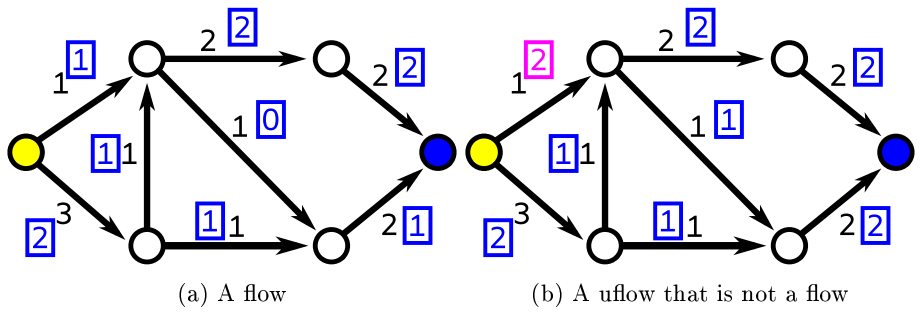


Figure 39: Both of these satisfy the Conservation Constraint at all nodes.

If a function takes a load function (43) as an argument, it can also take a flow (46) or a uflow (47), using their internal load function (either L_U or L_F).

Lemma 3. *If and only if a uflow (47) U on a network N has the property $Ut(U, N) \leq 1$, then U is also a flow (46).*

Proof. Begin by noting that both uflows and flows both are ordered triples of load function, source node, and terminal node. Additionally, both the load function of a flow and a uflow must follow the conservation constraint, so the only difference is the capacity condition.

If and only if $Ut(U, N) \leq 1$ is L_U feasible (44) then the capacity condition is satisfied. □

If something is defined on flows, it is also defined for uflows equivalently and vice versa if the capacity condition is satisfied.

Proposition-Definition 4. *For a uflow (47), U, $\forall x \in \mathbb{Q}^+, x \cdot U = U' = (x \cdot L_U(a), s(U), t(U))$ is a uflow.*

Proof. Since $x \in \mathbb{Q}^+$, $x \cdot L_U$ is guaranteed to be a load function, all that's needed is the conservation condition.

Let $I = V(N) - \{s(U), t(U)\}$, and replace $L_U(a)$ with $x \cdot L_U(a)$ in the conservation condition, to show that $L_{U'}(a)$ is a uflow's load function.

$$\forall n \in I \to \sum_{a \in A^-(n)} r \cdot L_U(a) - \sum_{a \in A^+(n)} r \cdot L_U(a) = 0$$

$$\forall n \in I \to r \cdot \sum_{a \in A^-(n)} L_U(a) - r \cdot \sum_{a \in A^+(n)} L_U(a) = 0$$

$$\forall n \in I \to r \cdot \left(\sum_{a \in A^-(n)} L_U(a) - \sum_{a \in A^+(n)} L_U(a) \right) = 0$$

$$\forall n \in I \to r \cdot (0) = 0$$

□

Proposition-Definition 5. *If two uflows (47), U_1 and U_2 share the same start and terminal nodes, s and t, then $U_1 + U_2 = U' = (L_{U_1}(a) + L_{U_2}(a), s, t)$ is a uflow and $\langle U' \rangle = \langle U_1 \rangle + \langle U_2 \rangle$.*

Proof. Since L_{U_1} and L_{U_2} , are load functions, their sum is guaranteed to be a load function, all that's needed is the conservation condition.

Let $I = V(N) - \{s, t\}$, and replace $L_U(a)$ with $L_{U_1}(a) + L_{U_2}(a)$, to show that $L_{U'}(a)$ is a uflow's load function.

$$\forall n \in I \to \sum_{a \in A^-(n)} L_{U_1}(a) + L_{U_2}(a) - \sum_{a \in A^+(n)} L_{U_1}(a) + L_{U_2}(a) = 0$$

$$\forall n \in I \to \left(\sum_{a \in A^-(n)} L_{U_1}(a) - \sum_{a \in A^+(n)} L_{U_1}(a) \right) + \left(\sum_{a \in A^-(n)} L_{U_2}(a) - \sum_{a \in A^+(n)} L_{U_2}(a) \right) = 0$$

$$\forall n \in I \to (0) + (0) = 0$$

Notice that also, since the value of a uflow is $\langle U \rangle = \sum_{a \in A^-(s(U))} L_U(a) - \sum_{a \in A^+(s(U))} L_U(a)$, or just $\langle U \rangle = \sum_{a \in A^-(s(U))} L_U(a)$ if no flow enters the sink, then, by commutativity of addition, $\sum_{a \in A^-(s(U))} L_{U'}(a) = \sum_{a \in A^-(s(U))} L_{U_1}(a) + L_{U_2}(a) = \sum_{a \in A^-(s(U))} L_{U_1}(a) + \sum_{a \in A^-(s(U))} L_{U_2}(a)$ Finally, notice that the same argument applies to the negative part, so $\langle U' \rangle = \langle U_1 \rangle + \langle U_2 \rangle$ ■ □

Subtraction is only defined if the difference in the load functions is non negative everywhere and the uflows have the same source and sink.

Definition 48. *A path flow (or path uflow, if the capacity condition is not satisfied) is a flow (46), F, on a network (42) N, where the set of arcs "used" (where the load function is positive) makes a dipath (38) from source to sink.*

It is easy to see that the conservation constraint requires all the positive arcs to have the same load across them. It also is clear to see the only uflows that can be subtracted from a path uflow are scaled versions of itself.

Definition 49. *A circulation is a flow (or uflow) (46), F for which $\langle F \rangle = 0$ and there's at least one arc, a such that $L_F(a) > 0$.*

Circulations, and uflows that a circulation can be subtracted from, do not follow all the same properties as ones a circulation can't be removed from. These uflows where a circulation can't be subtracted are called "circulation free" and can be made by adding together path uflows (48). This can be seen because the only thing that can be subtracted from a path uflow is a scaled version of itself, and there's no way to sum path uflows so that the value is zero.

A circulation is also the only kind of uflow where the source and sink are irrelevant, as $\langle U \rangle = 0$ is equivalent to the Conservation Condition applying on the source and sink.

Example 28. *First, begin with a circulation on its own.*

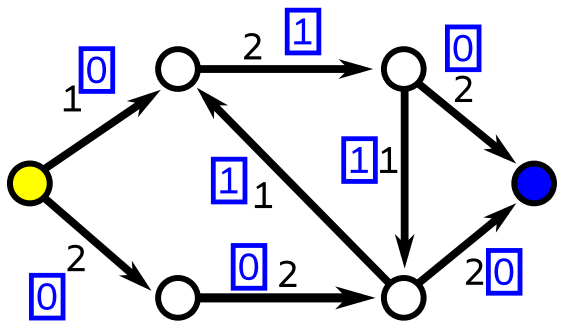


Figure 40: Notice how no flow is entering or leaving, but it is not zero everywhere.

Then, look at the following flow, and notice that the flow's load function at every arc is bigger than or equal to the circulation. This means that this flow "contains" a circulation.

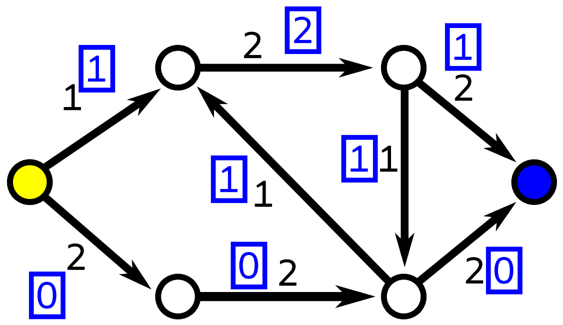


Figure 41: While the flow has value one, it has an arc with two flow across it.

If you didn't already know that a uflow had a circulation in it, how might you find out? While in general this is hard, it is obvious if an arc has more in it than the value of the uflow.

3.4 Value and Utilization

In problems involving flows, finding ones with a particular value, or the maximum value, is always an important consideration. For this paper, seeing how the value and utilization interact is a primary concern.

First, begin with our goal, of a function that gives the maximum value of the flow.

Definition 50. *The Meta Value function,* $\mathbb{V} : N, V(N), V(N) \rightarrow \mathbb{Q}$*, takes a network, a source, and a sink, and returns the maximum value of a flow (46)* *from the source to sink. If the source and sink are clear from the problem, then* $\mathbb{V}(N)$ *is acceptable.*

This has a value if the maximum flow on a network is a defined quantity, which the Max Flow Min Cut theorem defines.

Theorem 6. *The maximum value of a flow (46) F, on network (42) N, with source s and sink t, is determined by the minimum sum of the weight function across a minimal cutset (40) from s to t.*

The proof technique I will take to this well known result is, as far as I know, original, if a bit indirect. However, it will motivate Dinic's Algorithm, which is the maximal flow finding algorithm I will use in this paper.

It will be useful to have a function that returns the value of the minimal cutset, while the Max Flow Min Cut theorem hasn't been proven.

Definition 51. *The Minimal Cutset Function,* $\mathbb{M} : N, V(N), V(N) \to \mathbb{Q}$*, takes a network, a source, and a sink, and returns* $\min_{c \in Ct}\{w(c)\}$*, where Ct is the set of all minimal cutsets (40) from s to t. If the source and sink are clear from the problem, then* $\mathbb{M}(N)$ *is acceptable.*

Note that by Max Flow Min Cut (Thm 6), this is the same as the Meta Value Function (50), however when $\mathbb{M}(N)$ is used, it reflects that this does not rely on this fact.

Lemma 7 (Zero Case of Max Flow Min Cut). *If for a network (42) N, source $s \in V(N)$, and sink $t \in V(N)$, so that* $\min_{c \in Ct}\{w(c)\} = 0$*, where Ct is the set of all minimal cutsets (40) from s to t, then the maximum value flow from s to t is zero.*

In other words, if $\mathbb{M}(N, s, t) = 0$ *(51), then* $\mathbb{V}(N, s, t) = 0$ *(50).*

Proof. Suppose, for the sake of contradiction, there existed a flow (46) F, from s to t with positive value. Without loss of generality, assume the flow is circulation (49) free (this can be done as a circulation does not add to the value). This means that F must be the sum of at least one path flow (48).

First, consider the case where F is a path flow. This means there is a dipath (38), P, on which $L_F()$ is non-zero. However, by definition of minimal cutset (40), for every minimal cutset c, there exists exactly one arc, a, in the path P, such that $a \in c$. Since the capacity of arcs in a network is non negative, $w(a) \leq w(c)$. By the capacity condition, $L_F(a) \leq w(a) \leq w(c)$. However, since $\min_{c \in Ct}\{w(c)\} = 0$, there exists at least one minimal cutset c such that $w(c) = 0$. Therefore, there exists an arc a in the dipath P such that $L_F(a) \leq w(a) \leq 0$. However, by the conservation constraint on a path flow, for any arc a on P, $L_F(a) = \langle F \rangle > 0$. Therefore, F can't exist.

Second, consider F as a sum of path flows. However, since above proved a path flow can't exist, a sum of them can't exist, so F with positive value does not exist, and $\mathbb{V}(N, s, t) = 0$. □

To prove the Max Flow Min Cut theorem (Thm 6) when the predicted maximum value is not zero, we will build up from path flows (48). This is done through the idea of a residual network, which is also quite useful for Dinic's Algorithm

Definition 52. *For a network (42) N, and a flow on that network (46) F, the residual network, $N - F$, is a network with the same underlying digraph as N with the capacity function being $w(a) - L_F(a)$.*

This leads to another useful theorem, the Residual Minimal Custet Theorem.

Theorem 8. *Let N be a network (42) with a circulation-free (49) flow (46), F, on it, with source s and terminus t. Then, if $N_r = N - F$, $\mathbb{M}(N_r) = \mathbb{M}(N) - \langle F \rangle$ (51).*

Note again that, by the Max Flow Min Cut theorem (Thm 6), the minimal cutset function (51),$\mathbb{M}(N)$, and the meta value function (50), $\mathbb{V}(N)$, are the same, it is just stated with Minimal Cutset as that's what the argument uses and it is used to prove this equality.

Proof. Note that there are two ways a flow can be circulation free: If it is a path flow (48) or it is the sum of path flows. Consider first the case where F is a path flow. This means that the sum of the load function on any minimal cutset c, $L_F(c) = \langle F \rangle$. Then, by definition of the Minimal Cutset Function (51) and residual (52),

$$\mathbb{M}(N_r) = \min_{c \in Ct}\{w(c) - L_F(c)\} = \min_{c \in Ct}\{w(c)\} - \langle F \rangle = \mathbb{M}(N) - \langle F \rangle$$

For the case where F is the sum of path flows, consider $F_1, F_2, \ldots, F_k$, as path flows that sum together to give F. By the argument above, $\mathbb{M}(N - F_1) = \mathbb{M}(N) - \langle F_1 \rangle$. This motivates a sequence of residual networks, where $N_0 = N$ and for $i \in \{1 \ldots k\}$, $N_i = N_{i-1} - F_i$. It is clear to see that $N_k = N_r$, and for any $i \in \{1 \ldots k\}$, $\mathbb{M}(N_i) = \mathbb{M}(N_{i-1}) - \langle F_i \rangle$. Since this is true for all N_i if $i > 0$, this can be repeated until $\mathbb{M}(N_0)$ is reached.

$$\mathbb{M}(N_k) = \mathbb{M}(N_0) - \sum_{i=1}^{k} \langle F_i \rangle = \mathbb{M}(N_0) - \left\langle \sum_{i=1}^{k} F_i \right\rangle = \mathbb{M}(N) - \langle F \rangle$$

□

Note that if flow F contains a circulation (49) equality is not preserved, so in general, $\mathbb{M}(N_r) \leq \mathbb{M}(N) - \langle F \rangle$, with equality only when F is circulation free.

Finally, we can use this to prove Max Flow Min Cut (Thm 6), restated below.

Theorem 9. *The maximum value of a flow (46) F, on network (42) N, with source s and sink t, is the same as $\min_{c \in Ct}\{w(c)\} = \mathbb{M}(N, s, t)$.*

Proof. First, assume that $\mathbb{M}(N) > 0$, as the Zero Case was covered in Lemma 7. This means that there is at least one path flow (48), F, exists on N. Let X be the largest rational number such that $X \cdot F$ is a flow on N, which makes it a "saturated" path flow. This number exists because, by definition of a path flow, $L_F()$ is not zero everywhere and by definition of a network, all capacities are finite.

Now, consider the network $N_r = N - X \cdot F$. Because F is circulation free, $\mathbb{M}(N_r) = \mathbb{M}(N) - \langle X \cdot F \rangle$ by the Residual Minimal Custet Theorem (Thm 8). If $\mathbb{M}(N_r) = 0$, then $\mathbb{V}(N_r) = 0$ by Lemma 7. Suppose, for the sake of contradiction, there exists a flow, F_2, with value greater than $\langle X \cdot F \rangle$. Then, $\mathbb{M}(N - F_2) = \mathbb{M}(N) - \langle F_2 \rangle < \mathbb{M}(N) - \langle X \cdot F \rangle = 0$ by RMC (Thm 8), which is a contradiction as $\mathbb{M}$ is the minimum of non negative quantities. Therefore, if $\mathbb{M}(N_r) = 0$, then $\mathbb{V}(N) = \langle X \cdot F \rangle = \mathbb{M}(N)$, as $X \cdot F$ is a flow on N, and any bigger flow would lead to a contradiction.

In the case where $\mathbb{M}(N_r) > 0$, notice that at least one more arc of N_r has capacity zero than N. Since N only has finitely many arcs to begin with and a network with all zero arcs has $\mathbb{M}(N) = 0$, this means the case where the residual is positive cannot continue forever. Let $N_0 = N$, $F_0 = X \cdot F$, and $N_1 = N_0 - F_0$. Then, for N_i until $\mathbb{M}(N_i) = 0$, F_i be a saturated path flow on N_i, and $N_{i+1} = N_i - F_i$. Define k to be the last index where $\mathbb{M}(N_k) > 0$. By RMC, (Thm 8), $\forall i \in \{0 \dots k\}, \mathbb{M}(N_{i+1}) = \mathbb{M}(N_i) - \langle F_i \rangle$. Rearrange this to $\langle F_i \rangle = \mathbb{M}(N_i) - \mathbb{M}(N_{i+1})$, then sum all of these equations from 0 to k.

$$\sum_{i=0}^{k} \langle F_i \rangle = \sum_{i=0}^{k} [\mathbb{M}(N_i) - \mathbb{M}(N_{i+1})] = \mathbb{M}(N_0) - \mathbb{M}(N_k) = \mathbb{M}(N)$$

Note that, by the properties of the value function, $\sum_{i=0}^{k} \langle F_i \rangle = \left\langle \sum_{i=0}^{k} F_i \right\rangle$. Define $F_\Sigma = \sum_{i=0}^{k} F_i$ for ease of notation, and use RMC (Thm 8) to get:

$$0 = \mathbb{M}(N_k) = \mathbb{M}(N) - \langle F_\Sigma \rangle \to \mathbb{M}(N) = \langle F_\Sigma \rangle$$

Again, consider proof by contradiction. If there existed a flow, F_{better}, such that $\langle F_{better} \rangle > \langle F_\Sigma \rangle$, then by RMC (Thm 8),

$$\mathbb{M}(N - F_{better}) = \mathbb{M}(N) - \langle F_{better} \rangle < \mathbb{M}(N) - \langle F_\Sigma \rangle = 0$$

So a better flow than F_Σ cannot exist on N, therefore it is a maximal flow, so $\mathbb{V}(N) = \langle F_\Sigma \rangle = \mathbb{M}(N)$ □

Example 29. *For a simple network, the fact a circulation-free (49) flow has the same value across every minimal cutset (40) can be verified directly. In the diagram below, each different color of oval marks inclusion in a minimal cutset. Notice how many there are, even in this simple example. To reduce clutter, I have only marked the value of the flow, not the capacity of the network.*

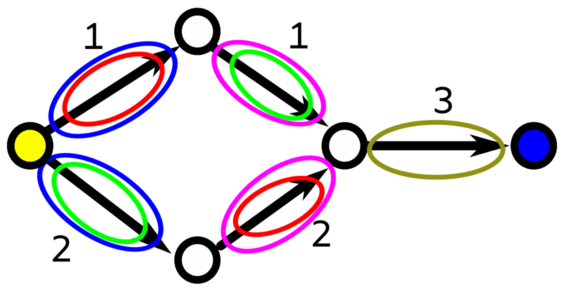


Figure 42: The sum over each cutset is three.

It is important that a flow contains a circulation may have cutsets with different values. Below, notice how the minimal cutset (40) marked with red arcs has a value of two, despite the flow only having a value of one. Since the arcs leaving the source are a minimal cutset here, the two minimal cutsets have different values.

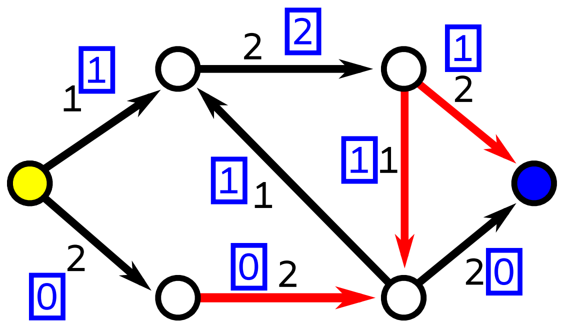


Figure 43: The red cutset is larger than the value.

3.4.1 Dinic's Algorithm

I cite [9] for the structure of how Dinic's Algorithm works.

Remember, that if a flow has a value, there is a circulation-free (49) flow with that same value. Hence, if a flow with maximum value ($\min_{c \in Ct}\{w(c)\}$) exists, there must be at least one max-value flow that has no circulations. If a flow has no circulations, then it is the sum of some set of path flows. Dinic's Algorithm adds these path flows together to reach a maximum flow.

Example 30. *Begin with the following simple network, with yellow source and blue sink.*

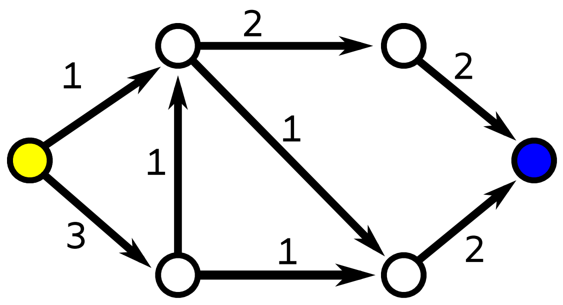


Figure 44: Our original network.

We begin by making a level network, where we start at the source, which we label with a zero, and then move along each arc, labeling the node we land on with a 1. Then, for each of those nodes, we go across all of their arcs, and if we encounter a new node, we label those nodes with a 2. If we reach a node we've already labeled, we color that arc red, to mark that it won't be in the level network. We continue like this until no node we've labeled can reach a node we haven't labeled.

Now, why is the level network important? On the level network, every walk (38) from source to sink is also a path. This is because, from any number n, the only arcs go to nodes labeled $n+1$. So, it is impossible to make a walk that isn't a path. It will be important later that these are also all the same length and the shortest path.

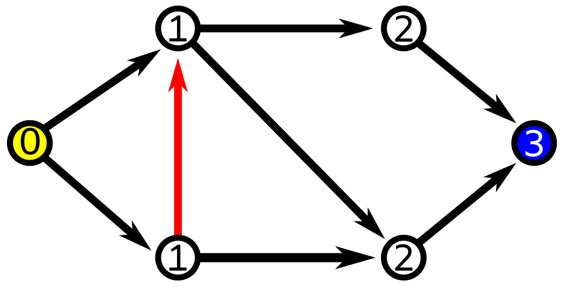


Figure 45: Our level network. Capacities are hidden for clarity.

In this example, only one arc is excluded. Next, find all the paths from source to sink on the level network.

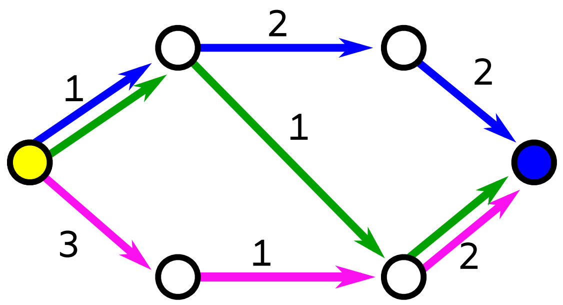


Figure 46: Paths on the level network, coded by color.

The goal is to chose a path, put the maximum flow you can over it, then pick another path flow to add until no more can be put on the level network This is called a "blocking flow", as it blocks the level network. It is important to note that the value of this flow does not need to be maximized. I pick the blue and pink to saturate, leaving me with the following flow after stage 1.

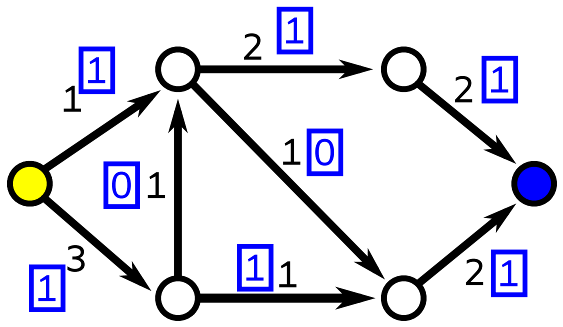


Figure 47: Note that we begin with the flow as zero everywhere.

Then, we subtract this flow from the network to get the residual (52) for the next step.

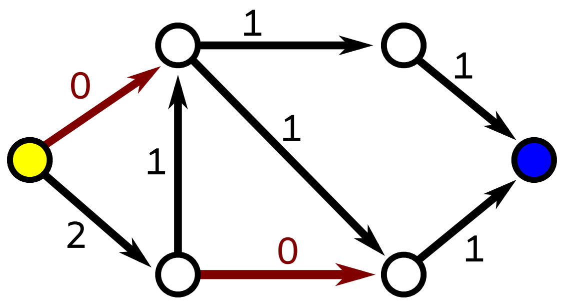


Figure 48: The arcs with 0 capacity are dark red as no flow can go across them.

The important thing to note here, before moving on to the next level network is that we treat any arc with capacity 0 as not existing.

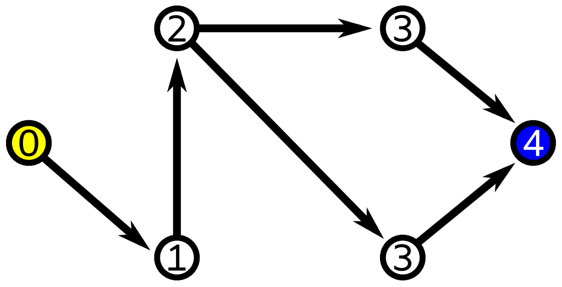


Figure 49: Our second level network. Capacities are hidden for clarity.

Notice that the sink is now further away, after removing the zero arcs. Since all of the non-zero arcs are in the level network, we can be sure there are no longer paths we're missing. Again, find the paths.

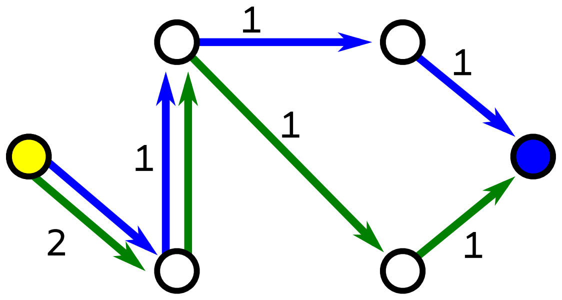


Figure 50: Paths on the level network, coded by color.

It is important to note that, while I am labeling all paths for clarity, it is not needed. All that is needed is to know when you have no more paths to find that aren't already blocked. In this case, only one path needs to be saturated to finish the method, and I picked blue. Combining putting the maximum flow (1) across the blue path with the flow from earlier results in the following flow.

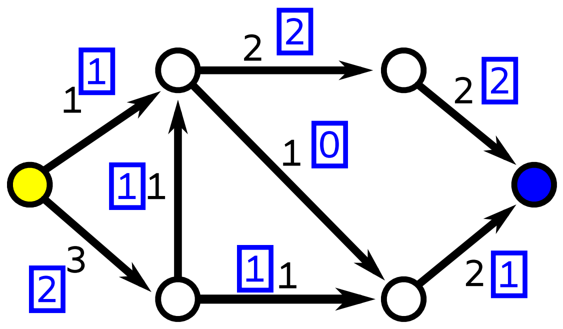


Figure 51: The final flow made up of paths.

We can check that this is really the last flow by trying to create a level network after removing that flow, and finding that we cannot reach the sink.

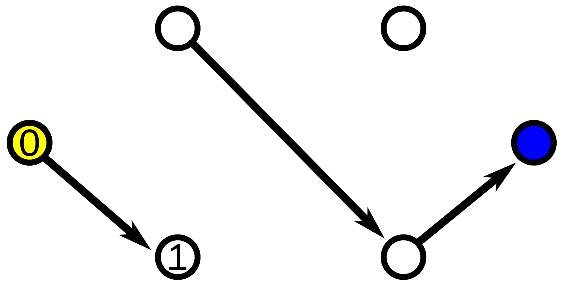


Figure 52: Notice how the sink is unlabeled, as there is no paths left.

In this case, I have constructed the example so it is easy enough to check that 3 really is the maximum flow by max flow min cut.

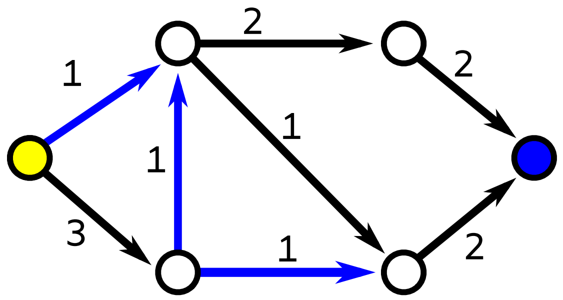


Figure 53: Blue arcs are in a cutset with total capacity 3.

Dinic's Algorithm: A Proof. First, note that there are only a finite number of paths from source to sink. This is because each path can visit each node at most once. Since there are finitely many paths, there are finitely many path lengths.

For each level network, all paths from source to sink are the same length. In fact, they must be the shortest paths on the residual network. This is because all of the nodes labeled 1 are reached with one move from the source, all labeled 2 are an arc away from a node labeled one, and so on, and there is no way to go backwards.

Once a blocking flow is found, by nature of being a blocking flow, there cannot exist other paths of that length on the level network. This means that the residual, after applying the blocking flow, also cannot have paths of that length. This is because any arc not included in the level network couldn't be part of a path of that length, since going along those does not get you further away from the source than you already were. Hence, the residual has (at least) one fewer path length than the prior network. Since there only finitely many path lengths, the method will always terminate.

By the proof of the Max Flow Min Cut theorem (9), if a sum of flows made from paths reduces the capacity of the minimal cutset to zero, then it must be a maximum value flow. □

If one network is larger than another where a flow is known, it will be useful to consider the new flow as "expanding" on the old one.

Theorem 10 (Expanded Network Theorem). *Consider a pair of networks, N_1 and N_2 with the same underlying digraph, and $\forall a \in A, w_2(a) \geq w_1(a)$. This means that N_2 is an "expansion" of N_1.*

If there is a flow, F_1, on network N_1, that is circulation-free (49), with source s and sink t, then there exists F_2 on network N_2 such that:

1. $\forall a \in A, L_{F_2}(a) \geq L_{F_1}(a)$
2. $\langle F_2 \rangle = \mathbb{V}(N_2)$
3. *F_2 has source s and sink t*
4. *F_2 is circulation free.*

Proof. Since $\forall a \in A, w_2(a) \geq w_1(a) \geq L_{F_1}(a)$, F_1 is a flow on N_2. Because of this, the residual (52), $N_3 = N_2 - F_1$ is defined.

Let F_3 be a circulation free flow on N_3 with source s, sink t, and maximum value. By the Residual Minimal Custet Theorem (Thm 8), $\langle F_3 \rangle = \mathbb{V}(N_2) - \langle F_1 \rangle$. Then, by addition of flows (5), $\langle F_1 + F_3 \rangle = \mathbb{V}(N_2)$. Since both F_1 and F_3 have source s and sink t, and are circulation free, their sum must be as well. Finally, since L_{F_3} is never negative, the sum $L_{F_1} + L_{F_3}$ is at least as big as L_{F_1}, therefore $F_1 + F_3 = F_2$. □

3.4.2 Max Flow/Value and Min Utilization Equivalence

To prove the connection between value and utilization, I'll introduce efficiency, a concept that links them together.

Definition 53. *The efficiency of a uflow (47) U on a network (42), given that $Ut(U,N) > 0$ (45), is $E(U,N) = \frac{\langle U\rangle}{Ut(U,N)}$*

It is trivial to see that if $\langle U\rangle > 0$, then $Ut(U,N) > 0$, so for all positive value uflows, it is defined and greater than zero. If a flow has value zero, it is defined to have efficiency zero, instead of doing the division.

Lemma 11. *Efficiency (53) is invariant to scaling of the uflow. Symbolically, $\forall x \in \mathbb{Q}^+, E(U,N) = E(x\cdot U,N)$.*

Proof. Begin by expressing the value of $\langle x\cdot U\rangle$ symbolically.

$$\langle x\cdot U\rangle = \sum_{a\in A^-(s(U))} x\cdot L_U(a) - \sum_{a\in A^+(s(U))} x\cdot L_U(a)$$

Factor out an x from the right hand side.

$$\langle x\cdot U\rangle = x\left(\sum_{a\in A^-(s(U))} L_U(a) - \sum_{a\in A^+(s(U))} L_U(a)\right)$$

$$\langle x\cdot U\rangle = x\,\langle U\rangle$$

Notice that by utilization property 3 (45),

$$Ut(x\cdot U,N) = x\cdot Ut(U,N)$$

Combining these together completes the proof.

$$E(x\cdot U,N) = \frac{\langle x\cdot U\rangle}{Ut(x\cdot U,N)} = \frac{x\,\langle U\rangle}{x\cdot Ut(U,N)} = \frac{\langle U\rangle}{Ut(U,N)} = E(U,N)$$

□

Theorem 12. *The flow (46) with the maximum value with source node S and terminal node T, on a network N (42), is a scaling of the uflow (47) with minimum utilization that has value one and the same source and sink and is on the same network.*

If there is no such uflow, then the maximum value is zero.

Proof. For the first case, assume that the set of uflows with source s, sink s, and value 1, called P, is not empty. Let $m = \min_{U\in P}\{Ut(U,N)\}$ and $U \in P : Ut(U,N) = m$. Then, since $\langle U\rangle = 1$, the efficiency (53) of U is $\frac{1}{m}$.

Because efficiency is invariant to the scaling of the uflow (by Lemma 11) and a uflow with utilization of 1 or less is a flow (by Lemma 3) $\frac{1}{m}\cdot U = F$ is a flow with utilization of one and the same efficiency. This means $\langle F\rangle = \frac{1}{m}$.

Assume for the sake of contradiction, there exists a flow $F' : \langle F'\rangle > \langle F\rangle$ between the same endpoints on N. Then, the efficiency of F', $E(F',N) =$

$\frac{\langle F' \rangle}{Ut(F',N)} \geq \langle F' \rangle > \frac{1}{m}$. By Lemma 11, $U' = \frac{1}{E(F',N)} \cdot F'$ has the same efficiency and value 1. However, this contradicts that $m = \min_{U \in P}\{Ut(U,N)\}$.

$$E(U',N) = \frac{1}{Ut(U',N)} = \frac{\langle F' \rangle}{Ut(F',N)} \geq \langle F' \rangle > \frac{1}{m}$$

$$\frac{1}{Ut(U',N)} > \frac{1}{m}$$

$$Ut(U',N) < m$$

Therefore, F must be the flow with the maximum value.

In case two, assume for the sake of contradiction there exists a flow F between s and t where $\langle F \rangle > 0$, and P (as defined before) is empty. This means that $E(F',N) \geq \langle F \rangle > 0$. By Lemma 11, $U = \frac{1}{E(F,N)} \cdot F$ has the same efficiency, and value of one. This contradicts that P is empty. Therefore, the maximum flow must be zero. □

This establishes a new way to think of "flow maximization" problems, and a re-contextualization of utilization minimization.

4 Obstructed Network Problem

While traditionally, the maximum flow is the goal for a single flow on a network, which is equivalent to a minimum utilization for a given value. However, one may encounter a situation where the network is obstructed in some way, perhaps by other flows. Then, finding a flow with minimum utilization *given the existing obstruction* can differ depending on the requested value.

Definition 54. *An obstructed network problem contains a network (42) N (which everything is on), an obstruction load function (43) L_O, a source s, sink t, and a goal value G for a flow between them.*

The objective is to find a uflow (47) U, $\langle U \rangle = G$, $s(U) = s$, and $t(U) = t$, that minimizes $Ut(L_O + U, N)$, letting U represent its load function.

4.1 Conversion to Affine Form

It turns out to be more convenient to work with a sum of networks for this kind of problem rather than its existing form. This is called "affine form" as the capacity of each arc on an "affine network" is an affine function of a "scaling variable". It will become clear later why this transformation is useful.

First, note that by definition of utilization (45), if $T = Ut(L_O + U, N)$ for some load function (43) L_O and uflow (47) U, then $Ut(L_O + U, TN) = 1$. This is then starting to look like a max flow problem, but that L_O being added makes it not quite the same. It would be nice if we could modify the problem to get U on its own.

Definition 55. *An affine network problem consists of two networks (42), N_b, N_s (the base network and scaled network), with the same underlying digraph, a source s, sink t, and a goal value G.*

The objective is to find the smallest $X \geq 0$, such that there exists a flow (46) F, from s to t, on the network $X \cdot N_s + N_b$, with value G. Since each weight in $X \cdot N_s + N_b$ is an affine function of X, the scaling variable, it is called an affine network problem.

It might be unclear how this kind of problem relates to the obstructed network problem we've been looking at in this subsection, but it turns out that every obstructed network problem has an equivalent affine network problem.

Theorem 13. *Any obstructed network problem (54) can be converted to an affine network problem (55), and the resulting X and F can be converted to the minimum utilization and uflow that gives that minimum utilization of the original problem.*

Proof. First, assume that N has no zero arcs (this can be done without loss of generality, since zero arcs can't have flow or load). Let $K = Ut(L_O, N)$. This means that, $\forall a \in A$, $\frac{L_O(a)}{w(a)} \leq K$. Multiply both sides by $w(a)$, $L_B(a) \leq K \cdot w(a)$, then subtract $L_O(a)$, to get $0 \leq K \cdot w(a) - L_O(a)$. Since the right hand side is never negative, it can be a capacity function of a network with the same underlying digraph as N. Hence, define $w_b(a) = K \cdot w(a) - L_O(a)$ as the capacity function for a network N_b.

Next, consider the goal of the problem, which is to find a uflow U, with source s and sink t, such that $\langle U \rangle = G$, and $T = Ut(L_O + L_U, N)$ is as small as possible. Use the definition of utilization to turn this into:

$$\forall a \in A, \frac{L_O(a) + L_U(a)}{w_N(a)} \leq T$$

Define $X = T - K$, to replace T with $X + K$, and multiply both sides by $w(a)$.

$$\forall a \in A, L_O(a) + L_U(a) \leq X \cdot w(a) + K \cdot w(a)$$

Subtract $L_O(a)$ from both sides, and note that $K \cdot w(a) - L_O(a) = w_b(a)$.

$$\forall a \in A, L_U(a) \leq X \cdot w(a) + w_b(a)$$

Finally, notice that this means that U is a flow on $X \cdot N + N_b$, with source s, sink t and value G. Since $T = X + K$, finding minimal X is the same as finding minimal T, and hence the problems are the same (after renaming N to N_s). $\square$

Note that for an affine network problem, if $\nexists a \in A : w_s(a) = 0$, which is the case for an affine network problem from an obstructed network problem, then $\forall \epsilon > 0, a \in A, X \geq 0, (X + \epsilon) w_s(a) + w_b(a) > X \cdot w_s(a) + w_b(a)$. This means that the value of a maximum flow on $(X + \epsilon) N_s + N_b$ is larger than the value of a maximum flow on $X \cdot N_s + N_b$.

Definition 56. *$V(X)$, for an affine network problem (55), is a function of a non-negative scaling parameter X, to the value on the maximum flown on $X \cdot N_s + N_b$.*

This can be defined also with the Meta Value Function (50) as

$$V(X) = \mathbb{V}(X \cdot N_s + N_b, s, t)$$

The goal is to find X such that $V(X) = G$ (if it is not the case that $V(0) \geq G$, as will be assumed going forward). The Max Flow Min Cut theorem (Thm 6) says that $V(X) = \min_{c \in Ct} \left\{ \sum_{a \in c} \left[X \cdot w_s(a) + w_b(a) \right] \right\}$, where Ct is the set of all minimal arc cutsets that separate s from t. This can be combined with $V(X) = G$ to obtain:

$$\forall c \in Ct, \sum_{a \in c} \left[X \cdot w_s(a) + w_b(a) \right] \geq G$$

$$\forall c \in Ct, X \sum_{a \in c} \left[w_s(a) \right] \geq G - \sum_{a \in c} \left[w_b(a) \right]$$

Assuming that w_s is zero nowhere, as can be assumed if the affine problem comes from an obstructed network problem:

$$\forall c \in Ct, X \geq \frac{G - \sum_{a \in c} \left[w_b(a) \right]}{\sum_{a \in c} \left[w_s(a) \right]}$$

$$X = \max_{c \in Ct} \left\{ \frac{G - \sum_{a \in c} \left[w_b(a) \right]}{\sum_{a \in c} \left[w_s(a) \right]} \right\}$$

This leads to an exact solution, but is very inefficient, as Ct is huge for all but the most trivial graphs.

4.2 The Derivative of the Value Function

In this paper, I will be using the right side derivative as the derivative, as it has some useful properties when working with flows. This means that the derivative of the value function (56), $V'(X)$ is defined as:

$$V'(X) = \lim_{\varepsilon \to 0^+} \left(\frac{V(X + \varepsilon) - V(X)}{\varepsilon} \right)$$

However, while this does have a value by the prior definition, a graph theoretic way of computation is not obvious.

To help with this, it is useful to define a function, $f(X)$, which takes in the scaling variable X and outputs a maximal, circulation free (49) flow on $X \cdot N_s + N_b$ with source s and sink t. Notice that, $\forall Y > X \geq 0$, $Y \cdot N_s + N_b$ is an "expanded network" compared to $X \cdot N_s + N_b$. By the Expanded Network Theorem (Thm 10), if F_X was a max flow on $X \cdot N_s + N_b$, then there would exist F_Y that follows all the conditions set by that theorem.

Definition 57. *For an affine network problem (55), the flow meta function, $f(X)$ is defined such that:*

1. *$\forall X \geq 0, f(X)$ is a max flow on $X \cdot N_s + N_b$ (with source s and sink t).*
2. *$\forall Y > X \geq 0, \forall a \in A, L_{f(Y)}(a) \geq L_{f(X)}(a)$*

Example 31. *First, consider the following affine network problem, looking at it as $X \cdot N_s + N_b$. This way, you can more easily see how the network will change as X increases.*

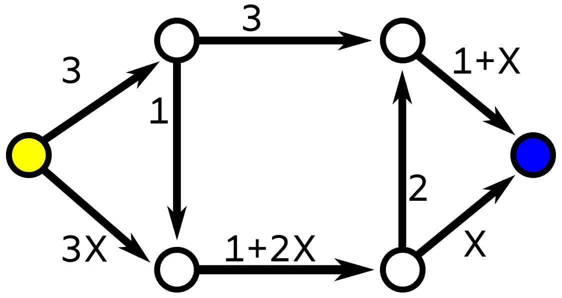


Figure 54: The sum of the networks for generic X, with yellow source and blue sink.

Note that the only value of X where the second condition of the flow meta function does not apply is at $X = 0$. So, any maximum flow on N_b is a valid $f(0)$. Here are two.

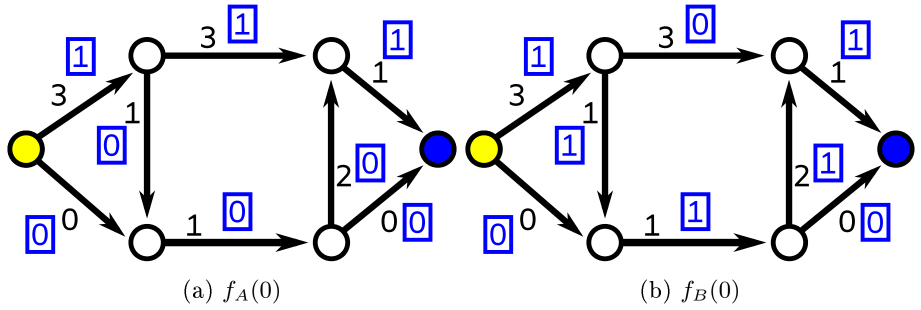


Figure 55: Both of these are valid beginnings to their own flow meta functions.

While $f_A(0)$ is the one you would find by Dinic's Algorithm (Ex 30), both of them are completely valid maximal flows. Consider then, what happens where $X = 1$ next. There's nothing inherently special about $X = 1$, we could pick any rational bigger than zero.

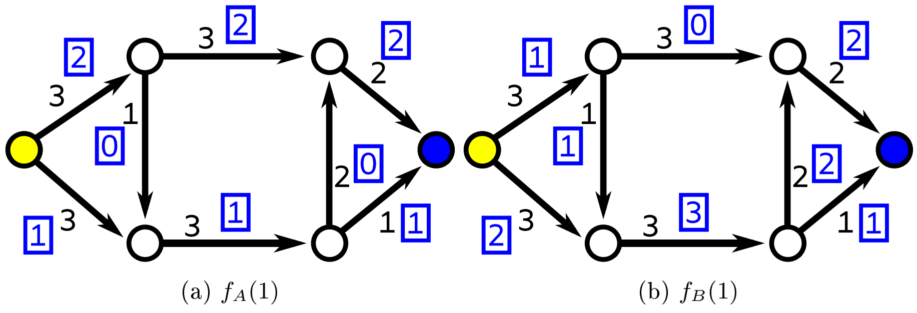
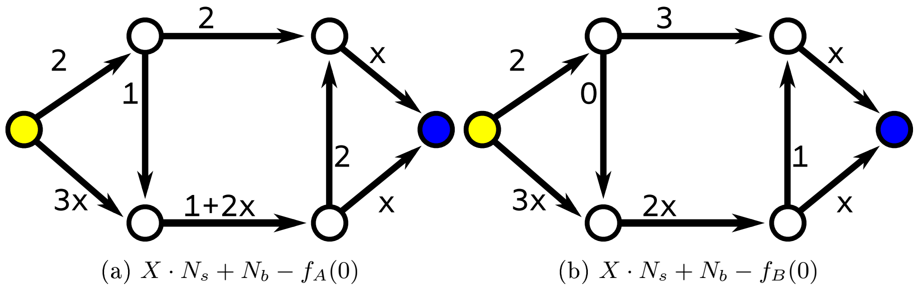


Figure 56: While $f_A(1)$ is a valid successor to $f_A(0)$, $f_B(1)$ is not and vice versa.

Take a look at how every arc in $f_A(1)$ has the same load or more than it had in $f_A(0)$, and the same for $f_B(1)$ and $f_B(0)$. But, importantly, there are arcs of $f_A(0)$ which are bigger than $f_B(1)$, and the same is true for $f_B(0)$ and $f_A(1)$.

To be sure we're computing valid successors, we can use a key step in the Expanded Network Theorem (Thm 10), and look at the residuals of $X \cdot N_s + N_b - f_A(0)$ and $X \cdot N_s + N_b - f_B(0)$. Then, realize that any flow on those, plus the corresponding $f(0)$ would be a valid $f(x)$.



2
2
x
1
2
3x
1+2x
x
(a) $X \cdot N_s + N_b - f_A(0)$
2
3
x
0
1
3x
2x
x
(b) $X \cdot N_s + N_b - f_B(0)$

Figure 57: While both of these residuals have the same max flow for every X, their flows are different.

These are useful if you have an $f(0)$ already, and wish to find a later entry. But, sometimes, you might not be interested in the flow meta function below a particular non-zero value. While these aren't truly "flow meta functions", it may be easier in some cases to not compute $f(0)$ first. As an example, consider $f_C(1)$ below.

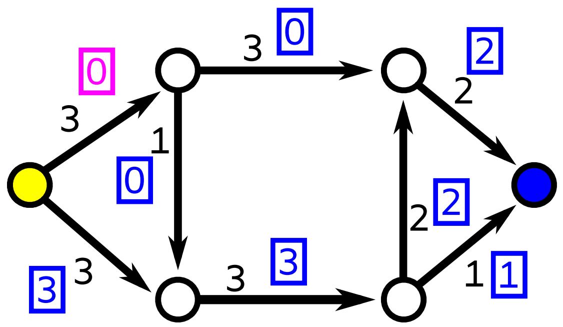
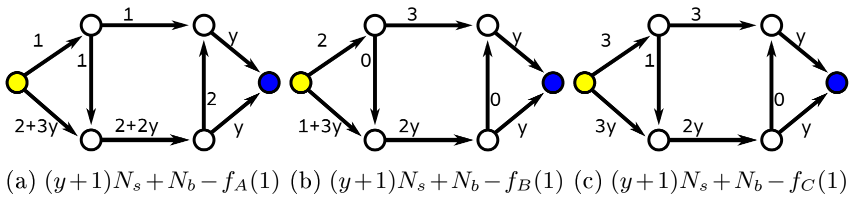


Figure 58: Note that the fuchsia load number is impossible if it is built off a max flow on N_b, as all flows on N_b that have non-zero value must use that arc.

Despite the fact this wouldn't be valid for a "true" flow meta function, if $X \geq 1$*, it can act like one. This can be called a "partial" flow meta function, where the lowest valid* X *is greater than zero. Usually, these aren't needed, as techniques that use flow meta functions tend to start with* $X = 0$*.*

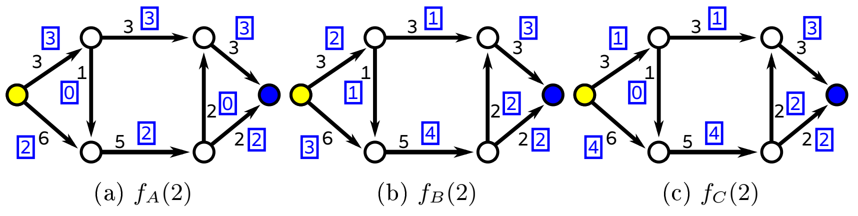
The residuals for the $f(0)$ *case are useful, but plugging that technique in directly for anything higher could lead to parts where the residual of* N_b *is negative. To correct this, let* $y = X - 1$*, and only consider* $y \geq 0$*. In general, the residual after* $f(T)$ *is* $(y+T)N_s + N_b - f(T)$*, where* $X = y + T$*. With this in mind, we can compute the following residuals:*



(a) $(y+1)N_s+N_b-f_A(1)$ (b) $(y+1)N_s+N_b-f_B(1)$ (c) $(y+1)N_s+N_b-f_C(1)$

Figure 59: These are valid, as the constant part is never negative.

From that, we can find $f_A(2)$*,*$f_B(2)$*, and* $f_C(2)$*. Notice how the change in value between this and* $f(1)$ *is less than the change between* $f(1)$ *and* $f(0)$*. This reflects that the flow has become more constrained by* N_b *now.*



(a) $f_A(2)$ (b) $f_B(2)$ (c) $f_C(2)$

Figure 60: Each is a valid successor to their own line, but not to any of the others.

Finally, remember that it is defined over all the positive rationals (or all rationals greater than or equal to 1 in f_C's case), not just the integers shown here.

With this, comes an alternate definition of the value function (56) as $V(X) = \langle f(X) \rangle$. From this we get $V'(X) = \lim_{\varepsilon \to 0^+} \left(\frac{\langle f(X+\varepsilon)\rangle - \langle f(X)\rangle}{\varepsilon} \right)$ which at first doesn't look much better until one realizes that, by property 2 of the Flow Meta Function (57) and the Expanded Network Theorem (Thm 10), one can define $f(X+\varepsilon)$ as $f(X)+\varepsilon F(X,\varepsilon)$, where $\varepsilon F(X,\varepsilon)$ is a max flow on $(X+\varepsilon)N_s + N_b - f(X)$ and hence, $F(X,\varepsilon)$ is a max flow on $\frac{N_s+N_b-f(X)}{\varepsilon} + N_s$ This means that $V'(X) = \lim_{\varepsilon \to 0^+} \left(\frac{\langle \varepsilon F(X,\varepsilon)\rangle}{\varepsilon} \right)$, which simplifies down to $\lim_{\varepsilon \to 0^+}(\langle F(X,\varepsilon)\rangle)$. This is why I had the ε in the first place. This is only possible with the right side derivative, because the converse of the expanded network theorem is not true.

To prove this limit exist and then compute it, begin with defining a set of arcs, B which includes all arcs a for which $X \cdot w_s(a) + w_b(a) - L_{f(X)}(a) > 0$. Note that, since $f(X)$ is a max flow on $XN_s + N_b$, there exists at least one minimal cutset c, for which $B \cap c = \emptyset$. Let N_u, where u stands for "unit", be a network with the same underlying digraph as N_s with the weight/capacity function, $w_u(a) = \begin{cases} 1 \, if \; a \in B \\ 0 \, if \; a \notin B \end{cases}$. Let $w_q(a) = X \cdot w_s(a) + w_b(a) - L_{f(X)}(a)$ for notation ease.

Since the set of arcs in B is finite, one can pick $0 < K_L \leq K_H$ (K low and high) such that, $\forall a \in B, K_L \leq w_q(a) \leq K_H$. So, for any $\varepsilon > 0$, $\frac{K_H}{\varepsilon}N_u + N_s$ is a bigger network than $\frac{N_q}{\varepsilon} + N_s$, and $\frac{N_q}{\varepsilon} + N_s$ is a bigger network than $\frac{K_L}{\varepsilon}N_u + N_s$. Therefore,

$$\lim_{K \to \infty} (\mathbb{V}(KN_u + N_s)) = \lim_{\varepsilon \to 0^+} \left(\mathbb{V}\left(\frac{N_q}{\varepsilon} + N_s \right) \right)$$

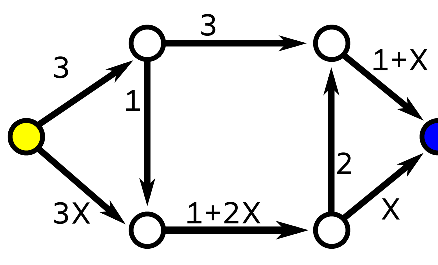
if the first limit exists and is finite by the squeeze theorem and the Expanded Network Theorem (Thm 10).

To prove the first limit is finite, first note that by the Expanded Network Theorem (Thm 10), if $K_2 > K_1$ then $\mathbb{V}(K_2N_u + N_s) \geq \mathbb{V}(K_1N_u + N_s)$. Then, by Max Flow Min Cut (Thm 6) and that there exists at least one minimal cutset c', where $w_u(c') = 0$, it must be true that $\forall K > 0, \mathbb{V}(KN_u + N_s) \leq w_s(c')$.

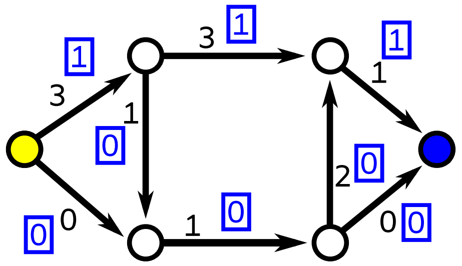
Consider letting $K = w_s(c')$. Then, $\mathbb{V}(KN_u + N_s) = \min_{c \in Ct}\{w_s(c')w_u(c) + w_s(c)\}$. Since $w_u(c)$ by definition can only be a non-negative integer for any set of arcs, $w_s(c')w_u(c) + w_s(c) \geq w_s(c')$ unless $w_u(c) = 0$. Therefore, any cutset with an element of B in it can be ignored, as they are at least as large as $w_s(c')$. Finally, note that this argument works for any $K > w_s(c')$ as well, so after $w_s(c')$, the limit is constant on a finite value.

Imagine then, solving the max flow problem on the "network" $\infty N_u + N_s$. Since the value of the flow is finite, the load on every arc is finite, so there exists a finite K such that the flow you found is a flow on $KN_u + N_s$. Then, there is a $\varepsilon > 0$ such that $\frac{N_q}{\varepsilon} + N_s$ is a bigger network than $KN_u + N_s$, so it is the limit, and hence $f'(X)$.

Example 32. *To compute the derivative of a flow meta function, we need both an affine network problem and a defined $f(X)$ for the derivative we want to compute.*



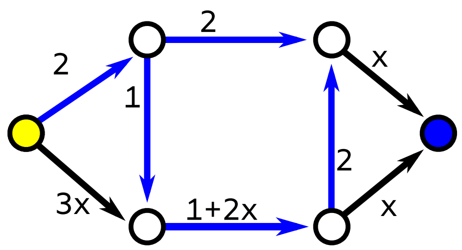
(a) The affine network



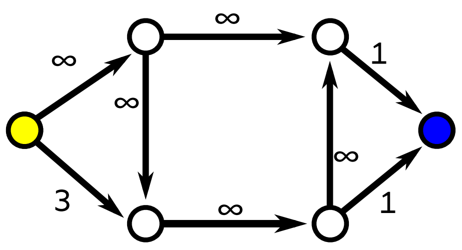
(b) $f(0)$

Figure 61: What we need to compute an $f'(0)$.

Note that there can be more than one $f'(X)$ for any X. These all will have the same value though, so for now, it does not matter which we compute. Next, look at the residual, highlight the elements of B (the arcs where $w_b(a) - L_{f(0)}(a) > 0$), and "make" $\infty N_u + N_s$.



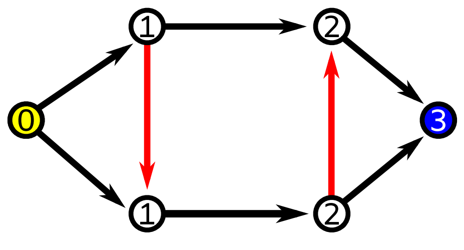
(a) Blue arcs are in B



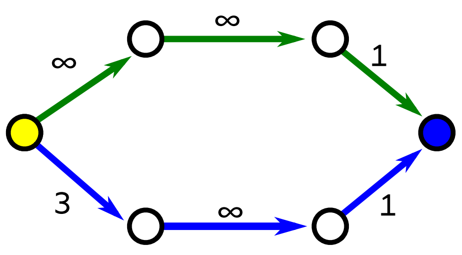
(b) The limit of $KN_u + N_s$ as $K \to \infty$

Figure 62: The second "network" does not truly exist, but is useful for computation.

Throwing caution to the wind, press on with Dinic's algorithm, to get the following level network and blocking flows.



(a) Our level network, red arcs are excluded, capacities hidden for clarity



(b) Each path has its own color

Figure 63: Remember that the maximum flow is 2, because of the cutset around the sink.

Since 1 across both of those paths is enough to block it, we're done immediately and our $f'(0)$ is:

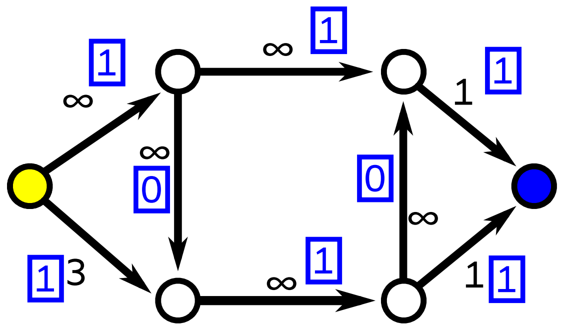


Figure 64: Shown on the infinite "network".

Again, realize that one could find other maximum flows on this infinite network, but all of them will have value 2.

Since the value function, $V(X)$ is non-decreasing and has non-increasing slope, since the minimum of affine functions must be convex, $\forall Y > X, V(Y) \leq V(X) + (Y - X)V'(X)$. Then by the definition of $f'(X)$, $V'(X) = \langle f'(X) \rangle$. So the value of $f(X) + \varepsilon f'(X)$ is $V(X) + \varepsilon V'(X)$. So, if $f(X) + \varepsilon f'(X)$ is a flow on $(X + \varepsilon)N_s + N_b$, then it must be a max value flow.

4.3 The Derivative Methods

With this derivative, the knowledge that $V(X)$ is made of a minimum of increasing affine terms, and a goal value, we can use this to find a way of getting to the answer or close to the answer faster than the cutset based "direct method". All of these methods are guaranteed to converge in finite time, but each has their own strengths and weaknesses.

4.3.1 Modified Bisection Method

Because $V(X)$ is non-decreasing (and in all cases from Obstructed Network Problems, increasing), the Bisection Method will work, but, as is common with the bisection method, there is no guarantee the exact answer will be reached. This can be shown by trying to find the inverse of $\frac{1}{3}$ with the function $f(x) = x$ on the range $[0, 1]$. Every guess by the bisection method must have a denominator that is a power of 2, and there is no fraction with a denominator that is a power of 2 that equals $\frac{1}{3}$.

With this in mind, how can one improve this searching method, with the properties of $V(X)$? A first thing to note is that, since the function is convex,$\forall z \in [X, Y], V(z) \geq \frac{V(Y)-V(X)}{Y-X}(z - X) + V(X)$, with equality when the slope is constant over the range. Since the slope of $V(X)$ is non-increasing, then $\forall z \geq X, V(z) \leq V'(X)(z - X) + V(X)$, with equality if $V'(z) = V'(X)$. So, from this, we can find a "low" and "high" estimate, x_L and x_H, where $V(x_L) \leq G$ and $V(x_H) \geq G$. Notice, that if the two estimates are the same, the slope over the range is constant, and hence, the solution is $x_L = x_H$. This means that if our range is narrowed down to two points with the same slope, the very next step will be the solution.

These two observations are key to the improvement to the generic Bisection Method that can be used in this case.

Suppose that we know $V(L) < G < V(H)$. Then, by the the properties of the value function before, $V(X_L) \leq G = V'(L)(X_L - L) + V(L) \therefore X_L = \frac{G-V(L)}{V'(L)} + L$ and then $V(X_H) \geq G = \frac{V(H)-V(L)}{H-L}(X_H - L) + V(L) \therefore X_H = \frac{(G-V(L))(H-L)}{V(H)-V(L)} + L$. Note that since the function is increasing, and $V(X_L) \leq G \leq V(X_H)$, the answer must be at the endpoints or in the middle. Average them to get, $X_M = \frac{X_L+X_H}{2} = \frac{G-V(L)}{2}\left(\frac{1}{V'(L)} + \frac{H-L}{V(H)-V(L)}\right) + L$, which cuts the range in half. Check $V(X_M)$. If it is less than G, then check $V(X_H)$. If that is greater than G, now you can have X_M, X_H as a new range to check. Likewise, if $V(X_M) > G$, then check $V(X_L)$. If that is less than G, your range is now X_L, X_M. If you find that one of them equals G, you are done.

Now, notice that if $V(X_L) < G$ at any step, then $\int_L^{X_L} V'(t)\,\mathrm{d}t < G - V(L)$. Dividing both sides by $X_L - L$ to get the average slope over the region results in the right hand side of the equation being $\frac{G-V(L)}{X_L-L}$, which given that $X_L = \frac{G-V(L)}{V'(L)} + L$, means this simplifies to $V'(L)$. But since the average must be less than $V'(L)$, and can never increase, at some point $P \in [L, X_L]$, $V'(P) < V'(L)$, so $V'(X_L) < V'(L)$. Finally, note that since the number of cutsets is finite, the number of possible slopes for the value function is finite, so the method must terminate.

However, it might not terminate quickly, so it's also useful to see how much smaller the range for the next step would be compared to the first step.

$$\frac{X_H - X_L}{H - L} = \frac{G - V(L)}{H - L}\left(\frac{H - L}{V(H) - V(L)} - \frac{1}{V'(L)}\right)$$

$$= \frac{G - V(L)}{V(H) - V(L)} \left(1 - \frac{V(H) - V(L)}{V'(L)\,(H - L)}\right) < \frac{G - V(L)}{V(H) - V(L)} < 1$$

Finally, note that since X_M is halfway between these, so the next steps range is half this value. So the range is improving more rapidly than with pure bisection.

The one loose end to tie up is how to find the initial L and H. Set $L = 0$. If $V(0) \geq G$, then that's the answer as the scaling constant can't be negative. Otherwise, continue. Note that $\forall X > 0, \mathbb{V}(XN_s) + V(0) \leq V(X)$, because $XN_s + N_b - f(0)$ is an expansion of network XN_S, so by the expanded network theorem, the max flow on the bigger network can't be smaller. So, since $\mathbb{V}(XN_s) = X\mathbb{V}(N_s)$ for any positive X, let $H = \frac{G-V(0)}{\mathbb{V}(N_s)}$, and this ensures that $V(H) \geq G$. Then, proceed on with finding X_L, X_H and X_M like any other step, or take $X_H = H$. Computing X_H in the first step does involve an additional flow calculation, but it narrows the search range more than letting it equal H. I do not know which is faster in general.

If F_s is a max flow on N_s, then $V(X) = X\langle F_s\rangle + V(0) + \mathbb{V}(X(N_s - F_s) + N_b - f(0))$. While this may look more complicated, the network $X(N_s - F_s) + N_b - f(0)$ may have more zero arcs on it, making the max flow calculation faster. In the example below, I will not use this as it makes the computation less clear, but for implementation, there's very little reason not to go for it.

Example 33. *First, examine our networks together, as in the prior example, combined together. The goal is to find the smallest X where a flow with value 20 exists on $XN_s + N_b$ between the yellow source and blue sink.*

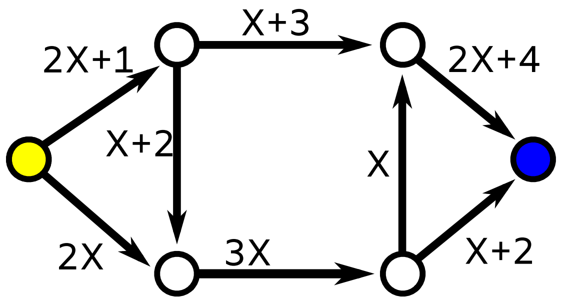


Figure 65: $XN_s + N_b$

I designed this example so that the answer is $X = 5$. If you want, compute the network where $X = 5$, and the maximum flow on it, to see that equals 20.

Since it is the first step, our $L = 0$. The first objective is to test the flow there. If it is 20 or above, the answer is $X = 0$.

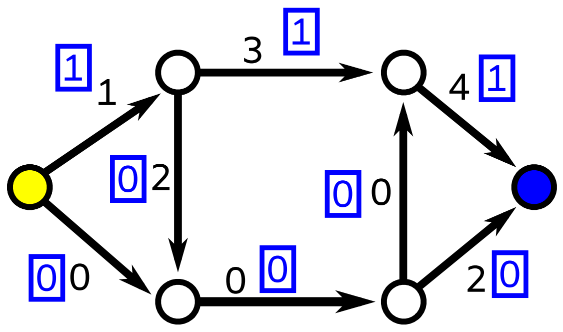


Figure 66: The maximum flow on N_b is 1.

Since this is less than 20, we continue by finding the maximum flow on N_s to allow us to compute our H.

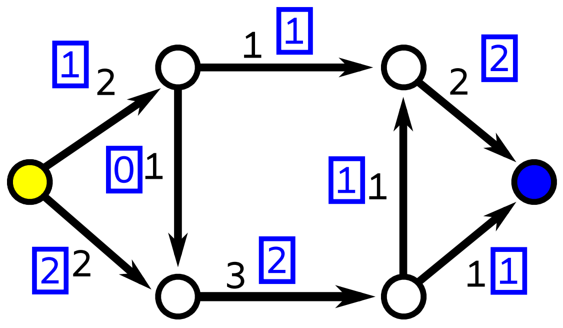


Figure 67: The maximum flow on N_s is 3.

Now, to get H, remember that $V(X) \geq V(0) + X\mathbb{V}(N_s)$. So, in this case, H must be such that $1 + 3H = 20 \leq V(H)$. This means $H = \frac{19}{3}$. For this example, let $X_H = H$ in the first step. This way, we have one fewer flow to compute.

To get X_L, we must find $V'(0)$. We can compute this by finding the maximum flow on an infinite "network", like in Ex 32.

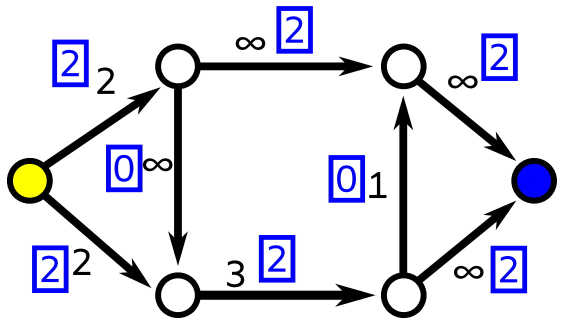


Figure 68: The maximum flow on this "infinite network" 4.

Notice that, despite all the infinite values, the smallest cutset, the one by the source, has the same sum as our value, so this is a max flow. Now, because $V(X)$ *is convex,* $V(X) \leq V(0) + XV'(0)$ *so* X_L *must be* $1 + 4X_L = 20 \leq V(X_L)$ *so* $X_L = \frac{19}{4}$.

Next, compute X_M. $X_M = \frac{X_L + X_H}{2} = \frac{19}{2}(\frac{1}{4} + \frac{1}{3}) = 5 + \frac{13}{24}$. *Notice how this is just a bit bigger than 5. And, if we didn't take the* $X_H = H$ *shortcut, our high would be lower, so* X_M *might be even closer. Try now to compute the maximum flow at* $5\frac{13}{24}$ *(all capacities and flow amounts are shown as mixed numbers).*

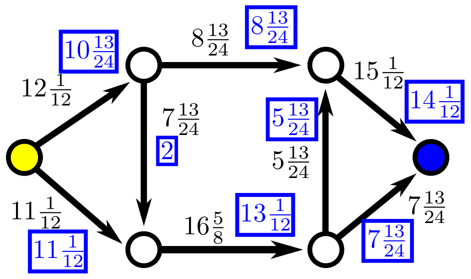


Figure 69: Since the value of $21\frac{5}{8}$ is larger than the goal, X_M is our new H.

Now we are in the second stage, where our bounds are $L = X_L = 4\frac{3}{4}$ *and* $H = X_M = 5\frac{13}{24}$. *First compute* $f(L)$, *then* $f'(L)$, *to get our* $V(L)$ *and* $V'(L)$ *respectively.*

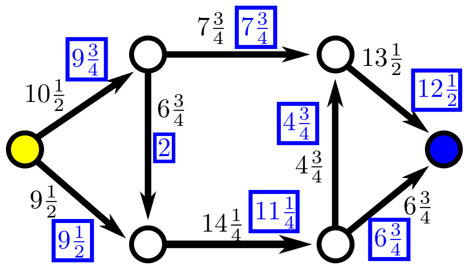


Figure 70: The value is $19\frac{1}{4}$, so the answer is greater than L.

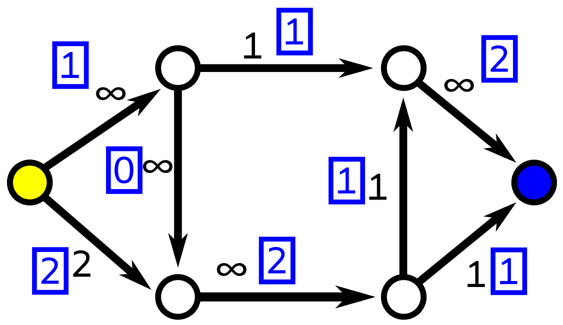


Figure 71: The slope of $V(X)$ has dropped to 3, the lowest possible slope.

We know that 3 is the lowest possible as the derivative network is an expanded network of N_s, so by the Expanded Network Theorem (Thm 10), the maximum flow on the derivative network is not less than the maximum flow on N_s.

Now, we can compute our new X_L and X_H.

$$X_L = \frac{G - V(L)}{V'(L)} + L = \frac{20 - 19\frac{1}{4}}{3} + 4\frac{3}{4} = \frac{3}{4 \cdot 3} + 4\frac{3}{4} = 5$$

We know that $X = 5$ is the actual answer, but let's continue with computing X_H as we would if we didn't know.

$$X_H = \frac{(G - V(L))(H - L)}{V(H) - V(L)} + L = \frac{(20 - 19\frac{1}{4})(5\frac{13}{24} - 4\frac{3}{4})}{21\frac{5}{8} - 19\frac{1}{4}} + 4\frac{3}{4}$$

Multiply the top and bottom of the left hand fraction by 8.

$$= \frac{(3)(11\frac{1}{12} - 9\frac{1}{2})}{19} + 4\frac{3}{4} = \frac{3(24 - 5)}{19 \cdot 12} + 4\frac{3}{4} = 5$$

Since both estimates are the same, our solution is $X = 5$.

4.3.2 Derivative Marching

While modified bisection is very useful for finding a particular X for a given goal G, it does not help if the goal changes but the networks do not. It also does not take advantage of the fact finding $V'(X)$ gives information about $V(X+\varepsilon)$ for ε sufficiently small. One way to map out the whole $f(X)$ metafunction is by "marching" along each $f'(X)$ found.

Start with $f(0)$, then find a $f'(0)$. It turns out that the $f'(0)$ we find may not be unique, as there can be several flows with the same value. Find the maximum value of ε_1 for which $f(0) + \varepsilon_1 f'(0)$ is a flow on $\varepsilon N_s + N_b$. Let $X_1 = \varepsilon_1 + 0$, as ε_1 is the amount we can move forward from our starting point of $X = 0$. Because the slope does not change on the range $X \in [0, X_1]$, then $V(X) = \langle f(0) \rangle + X \langle f'(0) \rangle$. If $\frac{G - V(0)}{V'(0)} \in [0, X_1]$, then $x = \frac{G - V(0)}{V'(0)}$ is the solution, and the flow is $f(0) + xf'(0)$. Otherwise, continue to the next step, with $N_1 = f(X_1) = f(0) + X_1 f'(0)$.

At step n, $f(X_n)$ is known, and it has value N_n. Find $f'(X_n)$, and find ε_{n+1}, the largest number such that for which $f(X_n) + (\varepsilon_{n+1})f'(X_n)$ is a flow on $(X_n + \varepsilon_{n+1})N_s + N_b$. Call $X_n + \varepsilon_{n+1} = X_{n+1}$ Then, $\forall X \in [X_n, X_{n+1}]$, $V(X) = N_n + (X - X_n)V'(X_n)$. If $x = \frac{G - V(X_n)}{V'(X_n)} + X_n$ is on the range $[X_n, X_{n+1}]$, this x is the solution. Otherwise, continue with the $n+1$ step.

If at any point, a $f'(X_n)$ is a flow on N_s, then the X_{n+1} is effectively infinite, and the method terminates. This is because for such a $f'(X_n)$, $\forall X > 0, a \in A, XL_{f'(X_n)}(a) \leq Xw_s(a)$. Add $L_{f(X_n)} \leq X_n w_s(a) + w_b(a)$ to both sides to get $XL_{f'(X_n)}(a) + L_{f(X_n)} \leq (X + X_n)w_s(a) + w_b(a)$ for all positive X. Letting $X_{n+1} = X + X_n$, we get $(X_{n+1})L_{f'(X_n)}(a) + L_{f(X_n)} \leq X_{n+1}w_s(a) + w_b(a)$ but this it true for any X_{n+1}, so for all $X > X_n$, $f(X) = f(X_n) + (X - X_n)f'(X_n)$ and hence the whole $f(x)$ function, and hence the whole $V(x)$ function, is known.

There are a few obvious weaknesses to this approach, such as that if G is large, this method will spend a lot of time marching through areas that are irrelevant for the purpose of finding G. This method also does not give an upper bound on X, only a lower bound. This means that early termination (that is, before the exact X is found) will not give good guesses. You can convert from a flow with value approximately equal to G to a uflow with value G for the original problem, but it will not be the most optimal. However, in cases where time is limited, re-scaling a flow with value almost equal to G is much better than returning nothing.

However, one less obvious weakness of this method is that it is possible for the bound on number of steps to reach the size of Ct, rather than the (potentially smaller) size of the number of slopes of $V(X)$. Note the use of the phrase "it is possible". This is because the particular $f'(X_n)$ can matter.

Example 34. *While all maximum flows on a network have the same value (by definition), this does not mean that marching using each of them is the same. Below, is the network we will consider. As always, yellow is the source, blue is the sink, and $XN_s + N_b$ is shown.*

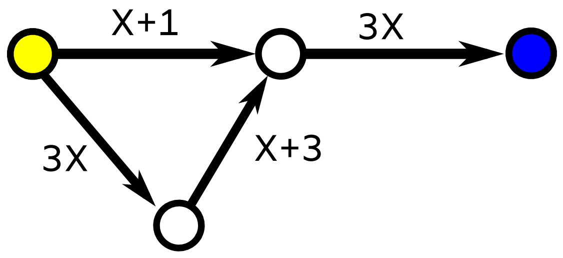
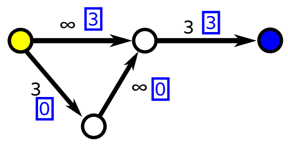
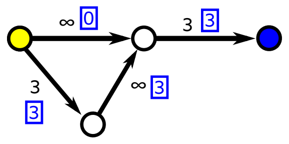


Figure 72: This is a very simple network to clarify visualization.

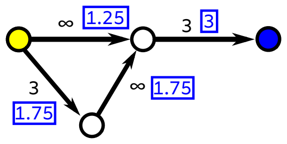
Notice, first of all, that the maximum flow when $X = 0$ has value zero, so $f(0)$ is zero on every arc. Then, consider the three following $f'(0)$. All of them are valid, as they block any more flow, but do so in different ways.



(a) This is the flow Dinic's Algorithm would make



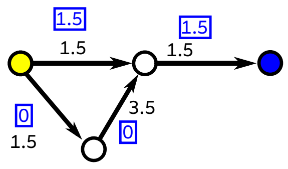
(b) This flow is one a "largest arc first" method would find



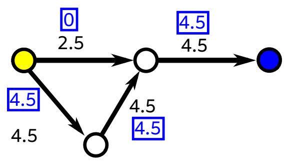
(c) This flow seems to be found arbitrarily

Figure 73: All three of these are maximum flows, and can be considered $f'(0)$.

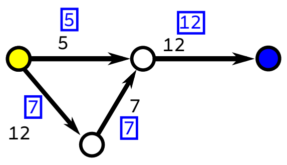
Now, applying the "marching" step, the first one can go to $X = \frac{1}{2}$, the second to $X = \frac{3}{2}$, and the last works until $X = 4$. For each flow, notice that they are a maximum flow, of the form $Xf'(0)$, for their own X value.



(a) Value: $1\frac{1}{2}$



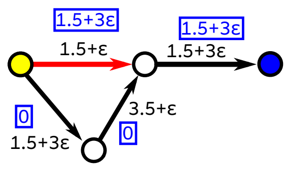
(b) Value: $4\frac{1}{2}$



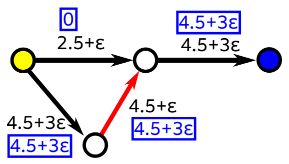
(c) Value: 12

Figure 74: Each of these flows are shown at largest scaling where they are a flow.

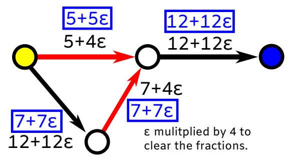
Next, why are these the largest they can go? Consider $(X+\varepsilon)f'(0)$ on the network $N_b + (X+\varepsilon)N_s$. Note that there's now too much flow on the red arcs.



(a) Note how the red arc needs to be 2ε bigger.



(b) Note how the red arc needs to be 2ε bigger.



(c) Epsilon has been multiplied by 4 to clear the fractions

Figure 75: Each of these are no longer a flow.

This proves that each of them can only go so far before a new derivative is calculated.

Then, the question is, how would you find these best derivative flows in general? Unfortunately, I do not know of a good method. I computed the third example by seeing what combination of the two paths was best, but this is relies on already knowing the optimal solution.

However, note that the bad f' still saturated a cutset of $XN_s + N_b$ for its maximum X, and this is guaranteed for any f'. This guarantee is because if it didn't, it wouldn't be at the maximum X for it. So, there can't be more steps than minimal cutsets.

4.3.3 Marched Search

While Derivative Marching is not an ideal method for many cases, it does have some computational advantages, such as how each step only involves solving one flow problem, while the Modified Bisection Method requires three (value at L or H (depending on which is known), slope at L, value at X_M) per step. It would be nice if you could know which one was best for the problem ahead of calculation.

Begin with a state with known L and H, where $V(L) < G < V(H)$, like in a middle step of the MBM. Calculate X_L, X_H, X_M like normal, but also calculate X_s, the "step X", which is the maximum value for which $f(L) + (X_s - L)f'(L)$ is a flow on $X_sN_s + N_b$. If $X_s \geq X_L$, then X_L is the answer, as the slope is constant on the range $[L, X_s]$. Otherwise, test $V(X_M)$. If this value is less than or equal to G, then proceed as normal, either terminating or setting the new range to X_M, X_H. However, if $V(X_M) > G$, then make the test range X_s, X_M. While this is a larger range, it has the advantage that $f(X_s)$ is already known, so the next step only requires two flow problems to be solved: finding $f'(X_s)$ and $V(X'_M)$ where X'_M is the X_M of the next step.

This marching can even be used to improve the bounds on the first step, since $\forall X \geq X_s$, $V(X) \geq V(X_s) + (X - X_s)\mathbb{V}(N_s)$, which is a bigger value than $V(0) + X\mathbb{V}(N_s)$ as $X_s\mathbb{V}(N_s) + V(0) < V(X_s)$. So the initial $X_H = \frac{G-V(X_s)}{\mathbb{V}(N_s)} + X_s$ is smaller than $\frac{G-V(0)}{\mathbb{V}(N_s)}$ which is used in MBM.

Of course, it might be also useful to sometimes make a "judgment call" on if X_s or X_L would be more efficient for a lower bound, but that is something I have not tested, and will leave up to the reader if there is a way to "tell" which one is more efficient in a particular step. Since computing X_s is only $O(A)$ in complexity, and can potentially avoid computing a flow or potentially solving the problem a step early (if $X_s \geq X_L$), which are on the order of $O(V^2A)$, it is worth considering.

5 Multi-Flow Problems

All flow problems up to this point have involved a single pair of source and sink. These are called single flow problems, and are much easier than the topic the paper has been leading up to, a particular multi-flow problem.

In a multi-flow problem, there is more than one pair of source and sink, and the **sum** of the load functions for each flow must be kept below the capacity of the arcs of the network. If one flow is maximized, another may be blocked, which leads to significantly more complexity. This kind of "blocking" should remind you of the Obstructed Network Problem, but in this case, the obstructions are other flows.

5.1 Topic of Paper

The primary question of this paper, sometimes referred to as "Load Balancing", has k flows to balance.

Given, on a network N, sequences of k sources $(s_1, s_2, \ldots, s_k)$, k sinks $(t_1, t_2, \ldots, t_k)$, and k goal values $(G_1, G_2, \ldots, G_k)$, find a sequence of k uflows, $U_1, U_2, \ldots, U_k$, where U_i has source s_i, sink t_i, and value G_i, such that the utilization of their sum is as small as possible.

A physical interpretation of this kind of optimization can be taken by considering a factory, where different components need to reach different parts of the factory in a particular ratio. Minimizing the utilization then, is ensuring that one unit can be made as quickly as possible.

5.1.1 K=1

First, let's consider the case where there is only one request. Let N be a network (42), s be the source node, t be the sink node, and $G > 0$ be the goal value. Notice that if $G = 1$, by Thm 12, this is equivalent to finding a maximum flow on N, then scaling the uflow to have value 1.

If $G \neq 1$, then consider the network $N' = \frac{1}{G} \cdot N$. On this network, find a maximum flow (call it F'), and use Thm 12 to convert it to the uflow with minimum utilization and value 1. Call this uflow U'. Then, set $U = G \cdot U'$. It is true that $Ut(U, N) = Ut(U', N')$, $\langle U \rangle = G$, and U' is the uflow with minimum utilization on N' with value 1. Hence, there doesn't exist a uflow with value G on N with a lower utilization.

Finally, note that since F' is a flow on N', then $F = G \cdot F'$ is a flow on N. Since F' is a maximal flow on N', $\nexists f' : \langle f' \rangle > \langle F' \rangle$ (with f' on N', and between the same endpoints). Imagine, for sake of contradiction, that f is a flow on N, where $\langle f \rangle > \langle F \rangle$. Scale the flow f and network N by $\frac{1}{G}$, and this statement leads to a contradiction that F' is a maximal flow on N'. Therefore, F is a maximal flow on N, and hence $U = \frac{G}{\langle F \rangle} \cdot F$ is the solution.

5.1.2 K=2

So, how much harder is having two flows than one? Well, it can be... Quite a bit harder. A simple, four node network will show how hard it is.

Example 35. *Let's try to solve a general form for a particular network*

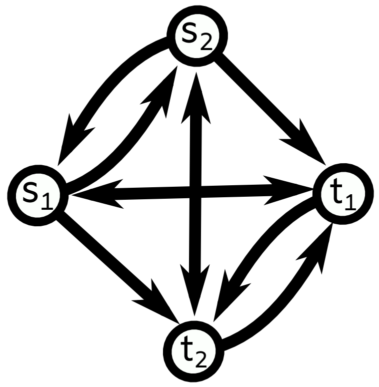


Figure 76: The Network I'll call N. The arcs can be described by the nodes they connect, as it is strict (29).

Now, consider all the dipaths from s_1 to t_1. All circulation free flows (49) are the sum of the flows across dipaths.

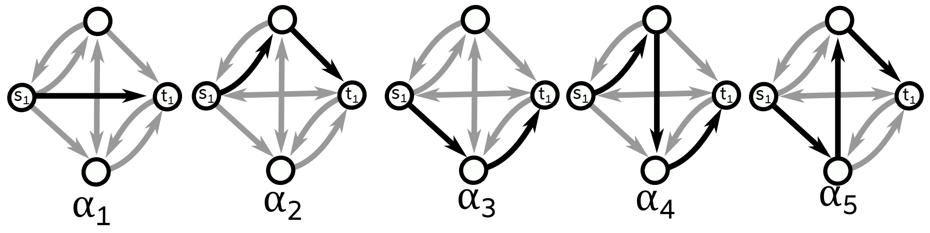


Figure 77: The Five paths from s_1 to t_1. We'll use those constants as the value on each path.

Given that definition of the α_i values, $\langle U_1 \rangle = \sum_{i=1}^{5} \alpha_i$. Next, do the same for s_2 and t_2

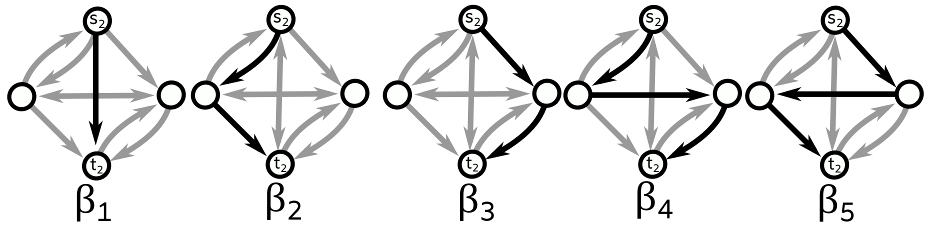


Figure 78: The Five paths from s_2 to t_2. We'll use those constants as the value on each path.

Again, by definition, $\langle U_2 \rangle = \sum_{i=1}^{5} \beta_i$.

Next, let's find the limits placed by the utilization:

$$\begin{aligned}
\alpha_1 + \beta_4 &\leq c(s_1,t_1)\cdot Ut\\
\beta_5 &\leq c(t_1,s_1)\cdot Ut\\
\alpha_4 + \beta_1 &\leq c(s_2,t_2)\cdot Ut\\
\alpha_5 &\leq c(t_2,s_2)\cdot Ut\\
\alpha_2 + \alpha_4 &\leq c(s_1,s_2)\cdot Ut\\
\beta_2 + \beta_4 &\leq c(s_2,s_1)\cdot Ut\\
\beta_3 + \beta_4 &\leq c(t_1,t_2)\cdot Ut\\
\alpha_3 + \alpha_4 &\leq c(t_2,t_1)\cdot Ut\\
\alpha_3 + \alpha_5 + \beta_2 + \beta_5 &\leq c(s_1,t_2)\cdot Ut\\
\alpha_2 + \alpha_5 + \beta_3 + \beta_5 &\leq c(s_2,t_1)\cdot Ut
\end{aligned}$$

This can be simplified to the following linear equation, letting $\rightharpoonup K$ *be some vector with entries between 0 and 1 inclusive and* $diag()$ *creating a matrix with the vector as the diagonal.*

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&1\\
0&0&0&1&0&1&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0\\
0&1&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&0&1&0\\
0&0&0&0&0&0&0&1&1&0\\
0&0&1&1&0&0&0&0&0&0\\
0&0&1&0&1&0&1&0&0&1\\
0&1&0&0&1&0&0&1&0&1
\end{bmatrix}
\begin{bmatrix}\vec{\alpha}\\ \hline \vec{\beta}\end{bmatrix} = Ut\cdot diag(\vec{K})
\begin{bmatrix}
c(s_1,t_1)\\ c(t_1,s_1)\\ c(s_2,t_2)\\ c(t_2,s_2)\\ c(s_1,s_2)\\ c(s_2,s_1)\\ c(t_1,t_2)\\ c(t_2,t_1)\\ c(s_1,t_2)\\ c(s_2,t_1)
\end{bmatrix}$$

For ease of notation, this can be represented as $M\begin{bmatrix}\vec{\alpha}\\ \hline \vec{\beta}\end{bmatrix} = Ut\left(diag(\vec{K})\vec{C}\right)$.

If M *was non-singular, we could reverse the equation to get the allowed alpha and beta vectors by trying different values for* $\vec{K}$*. This would look like:*

$$\begin{bmatrix}\vec{\alpha}\\ \hline \vec{\beta}\end{bmatrix} = Ut\left(M^{-1}\left(diag(\vec{K})\vec{C}\right)\right)$$

However, M *is a singular matrix, so this inversion is impossible.*

Since M is singular, there exists a non zero vector (and scalar multiplications of it) $\vec{z}$, such that $M\vec{z} = \vec{0}$. This $\vec{z} = \begin{bmatrix}1 & -1 & -1 & 1 & 0 & -1 & 1 & 1 & -1 & 0\end{bmatrix}^{\mathsf{T}}$

Hence, $M\left(\begin{bmatrix}\vec{\alpha}\\ \hline \vec{\beta}\end{bmatrix}\right) = M\left(\begin{bmatrix}\vec{\alpha}\\ \hline \vec{\beta}\end{bmatrix} - \alpha_1\vec{z}\right)$. Note that the vector multiplying the matrix, $\vec{v}$, has a zero first row, so as long as some matrix M' has all but the first column the same, $M\vec{v} = M'\vec{v}$. We can then pick an M' such that it is invertable. My choice for this M' is:

$$M' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Using the inverse of this matrix results in:

$$\vec{v} = \frac{Ut}{3} \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 \\ 3 & -3 & 0 & -3 & 1 & -1 & -2 & -1 & 1 & 2 \\ 3 & -3 & 0 & -3 & -1 & -2 & -1 & 1 & 2 & 1 \\ -3 & 3 & 0 & 3 & 1 & 2 & 1 & 2 & -2 & -1 \\ 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & -3 & 3 & -3 & -1 & -2 & -1 & -2 & 2 & 1 \\ -3 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & -1 & 1 & 2 & 1 & -1 & 1 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} (\vec{K} \otimes \vec{C})$$

Then, this can be simplified given that the top row must be zero, giving:

$$\vec{v} = Ut \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & 1 & 0 \\ -1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 1 & -1 & 0 & -1 & 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} (\vec{K} \otimes \vec{C})$$

Next, use T for that matrix, and turn $\vec{v}$ back into $\left[\begin{array}{c} \vec{\alpha} \\ \hline \vec{\beta} \end{array}\right]$.

$$\left[\begin{array}{c} \vec{\alpha} \\ \hline \vec{\beta} \end{array}\right] = Ut\,(T)\,(\vec{K} \otimes \vec{C}) + \alpha_1 \vec{z}$$

Next, set the goal for flow 1 to 1, and the goal of flow 2 to $g \in (0, 1]$. This makes the problem into finding $\vec{K}$ and $\alpha_1 \geq 0$ such that Ut is as small as possible, given that:

$$\begin{bmatrix} 1 \\ g \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \left(Ut\,(T)\,(\vec{K} \otimes \vec{C}) + \alpha_1 \vec{z} \right)$$

and $\left(Ut\,(T)\,(\vec{K} \otimes \vec{C}) + \alpha_1 \vec{z} \right)$ is negative nowhere and the top row of $T(\vec{K} \otimes \vec{C})$ is zero.

5.2 When is this easy?

despite the complete change over, I still feel like this is a distraction, but I'm giving it my best shot in this draft

Fortunately, there are times that some paths for a flow are "free", that is, they can be taken without obstructing any other flow. Likewise, there are pairs of paths which are "semi-free", where each path goes between a different pair of sources and sinks, and are the only overlap. For a semi-free pair, if one flow needs more than the other, it can take it, knowing it will only impact one other path.

With this in mind, it is clear that the example above has no free paths or semi-free pairs of paths, instead all being overlapped on top of each other. This makes it so taking any source to sink path impacts the other flow.

5.3 Bounds

Finally, before tackling my method for $K > 1$, starting with some easy bounds is a way to see what to expect.

Theorem 14. *An optimal solution for a list of sources $(s_1, s_2, \ldots, s_k)$, sinks $(t_1, t_2, \ldots, t_k)$ and goal values $(N_1, N_2, \ldots, N_k)$, on a network N, cannot have a higher utilization than $Ut(\sum_{i=1}^{K} U_i, N)$, where U_i is a maximum efficiency uflow from s_i to t_i with value N_i.*

Proof. Note that the optimal solution is the one with the lowest $Ut(\sum_{i=1}^{K} U_i, N)$ for some sequence of U_i. Hence, if the U_i as I defined it is a solution, an optimal solution is the same or better. Finally, note that $\langle U_i \rangle = G_i$, $s(U_i) = s_i$ and $t(U_i) = t_i$ for all $i \in \{1 \ldots K\}$. This means it is a solution and by definition of optimal, the optimal solution must be the same or better. $\square$

Theorem 15. *An optimal solution for a list of sources $(s_1, s_2, \ldots, s_k)$, sinks $(t_1, t_2, \ldots, t_k)$ and goal values $(G_1, G_2, \ldots, G_k)$, on a network N, cannot have a lower utilization than $\max_{i \in \{1 \ldots K\}} \{Ut(U_i, N)\}$, where U_i is a maximum efficiency uflow from s_i to t_i with value N_i.*

Proof. This will be a proof by contradiction. Let $\mathbb{T} = \max_{i \in \{1 \ldots K\}} \{Ut(U_i, N)\}$, with U_i defined as the theorem states.

Assume, for sake of contradiction, there exists a sequence U_i', where $Ut(\sum_{i=1}^K U_i', N) < \mathbb{T}$. By the Triangle Inequality for the utilization function (property 1),

$$\max_{i\in\{1\dots K\}}\{Ut(U_i', N)\} \leq Ut(\sum_{i=1}^{K} U_i', N) < \mathbb{T}$$

This means,

$$\forall i \in \{1\dots K\}, Ut(U_i', N) < \mathbb{T} = \max_{i\in\{1\dots K\}}\{Ut(U_i, N)\}$$

Choose i such that $Ut(U_i, N) = \mathbb{T}$. Then, $Ut(U_i', N) < Ut(U_i, N)$. But this contradicts that U_i is a maximum efficiency uflow with that source, sink, and value. So, the sequence U_i' cannot exist. $\square$

Example 36. *Consider the following network, where each source has a light color (and is numbered with which flow it belongs to) and each sink has a dark color (again, with numbers). The goal value for each flow is 6.*

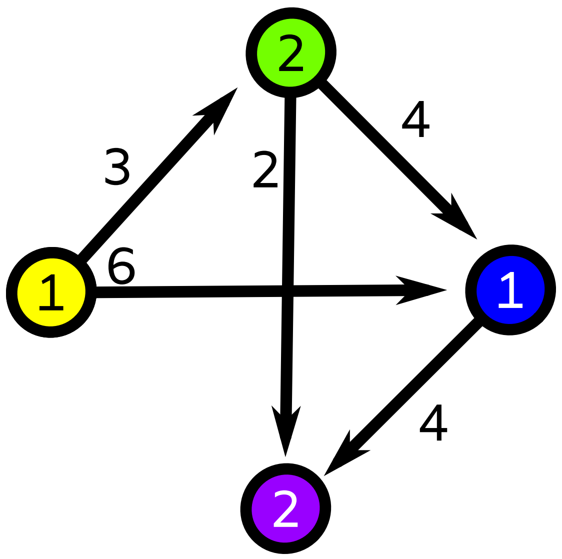
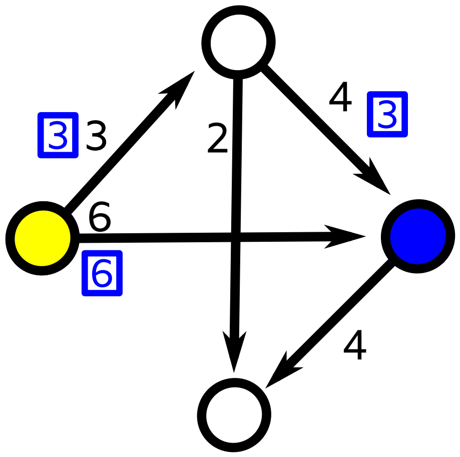
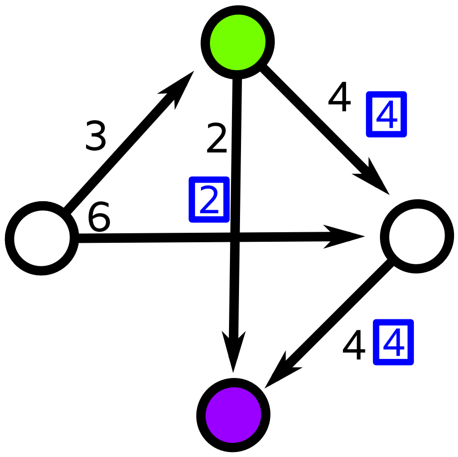


Figure 79: Notice that there are only two paths for each flow from source to sink.

Begin with the naive approach used with the upper bound, and find the maximum value for each flow.



(a) Since the value is 9, this flow is scaled by $\frac{2}{3}$



(b) This flow has value 6, so it is unchanged

Figure 80: The maximum flow for each pair of source and sink.

Then, add these flows together, scaling the first flow down to have a value of 6 instead of 9.

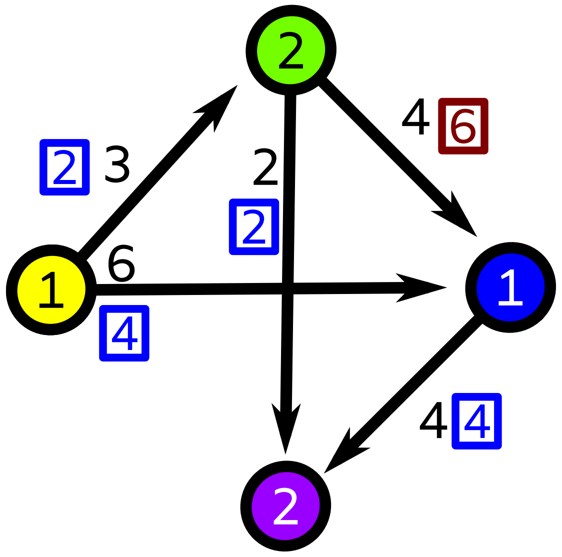


Figure 81: The dark red number is over capacity, making this solution have a utilization of 1.5.

However, taking the first flow and making it go only along the direct route improves the utilization to 1, as seen below.

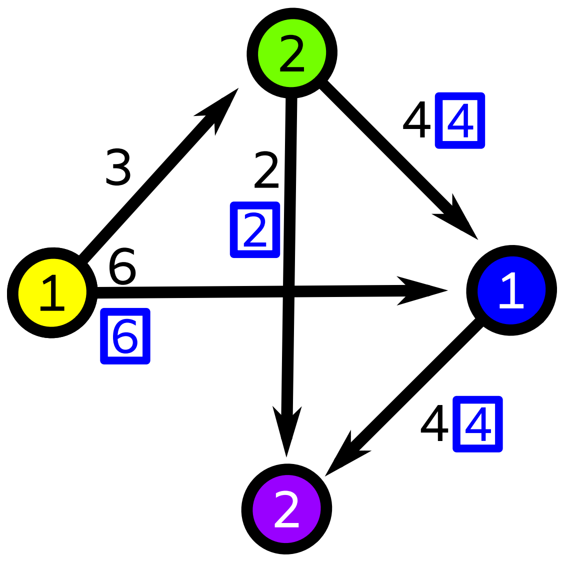


Figure 82: Now that the second flow only uses the direct path, the utilization is 1.

This cannot be improved further, by the proof of the lower bound (Thm 15).

There are also cases where the upper bound is the best possible. The most trivial version of this is a network where each pair is in a different component (22), as in the figure below.

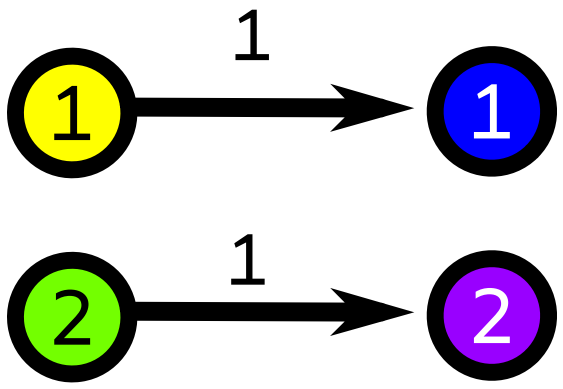


Figure 83: Whatever goals are demanded, both bounds will be $\frac{1}{\max\{G_1,G_2\}}$.

However, the upper bound can be the best possible in situations where the lower bound is smaller.

Example 37. *Consider the following network, with the same conventions as Ex 36. Again, the goal for each flow is 6.*

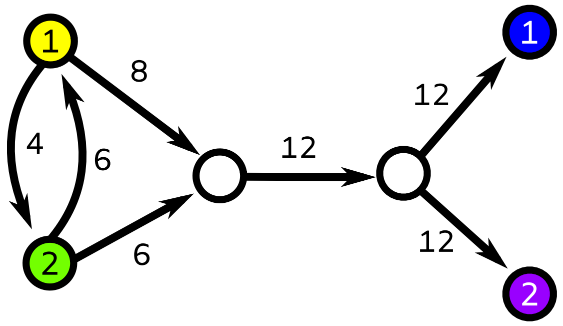
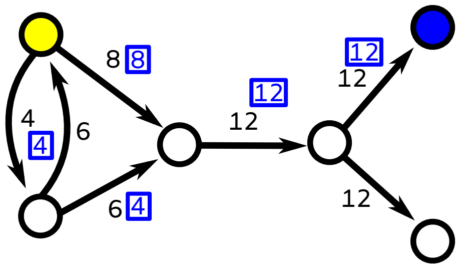
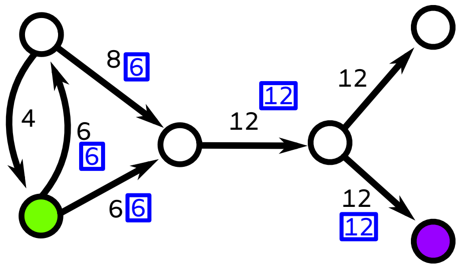


Figure 84: Notice the single arc that connects the left and right sides together.

Begin with finding the maximum value for each flow.



(a) Since the value is 12, this flow is scaled by $\frac{1}{2}$



(b) Since the value is 12, this flow is scaled by $\frac{1}{2}$

Figure 85: The maximum flow for each pair of source and sink.

Then, add these flows together, and scale down by half so they have a value of 6 instead of 12.

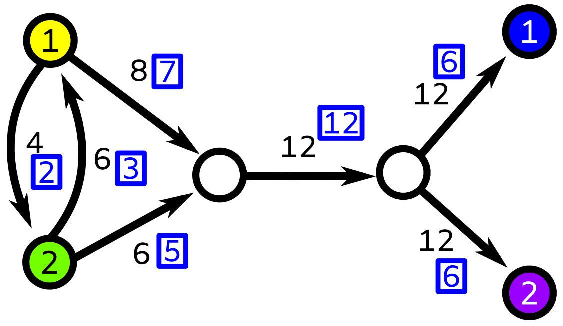


Figure 86: Notice that the arc connecting the two halves is fully saturated.

Because the middle arc is the only way to reach the sink from the source, there is no way to avoid having 12 flow across it. Therefore, the best possible *utilization is 1.*

5.4 Amulo's Flow-wise Fair Progressive Approximation Algorithm

The Algorithm I've devised for the Load Balancing problem, AFFPAA, approximates an optimal solution by progressing one flow at each step to lower the utilization. The algorithm is "fair" as at each step, each uflow has the requested value, no more, no less. This means at any "stage" (discussed later), it can be stopped to give an approximate answer, which is a clear advantage over a linear programming based approach, where stopping the algorithm early may result in load functions that are not uflows.

AFFPAA begins with computing the maximum value flow between each source and sink, then scaling them to a uflow with the goal value. These are labeled $U_1^{(0)}, U_2^{(0)}, \ldots, U_k^{(0)}$, where $U_i^{(0)}$ is the uflow from s_i to t_i with value G_i and minimum $Ut(U_i, N)$. The upper index in parenthesis is the "stage" along in the algorithm, and at the start of each stage is a solution to the problem, and has a lower total utilization than the start of the previous stage. Define $\mathbb{T}^{(m)}$ to be the total utilization at the start of a stage, which is $Ut(\sum_{i=1}^{k} U_i^{(m)}, N)$

Advancing a stage begins by conducting "trials". For each trial, pick $i \in \{1 \ldots k\}$. I will discuss different ways of picking this i later, just know that each i will be picked *at most* once. With the i picked, compute the sum of the load functions of every other uflow, $L_O = \sum_{j=1, j\neq i}^{k} L_{U_j^{(m)}}$ where m is the stage number. This is the "obstruction" for an obstructed network problem (54). Notice that the goal of an obstructed network problem, minimizing $Ut(L_O + U_O, N)$ (using U_O instead of U to designate it's the obstruction problem version), is the same as minimizing the total utilization, if U_O has source s_i, sink t_i, and goal value G_i. This means that U_O can "replace" $U_i^{(m)}$ in the next stage (if chosen).

The optimal U_O can be computed using any of the tools I discussed for solving the obstructed network problem in that section of the paper. Define the "trial utilization", $\mathbb{T}_i^{(m)}$, to be $Ut(L_O + U_O, N)$. Notice that $\mathbb{T}_i^{(m)} \leq \mathbb{T}^{(m)}$, as for $U_O = U_i^{(m)}$, $Ut(L_O + U_O, N) = \mathbb{T}^{(m)}$, so the best U_O can't be worse. If $\mathbb{T}_i^{(m)} = \mathbb{T}^{(m)}$, we consider the trial to be a failure, as replacing $U_i^{(m)}$ with U_O will not improve the total utilization. Otherwise, the trial is a success, and we "save" the U_O from this trial as $U'^{(m)}_i$, for later consideration.

If all possible trials end in failure, the algorithm terminates, as it cannot find a better solution from its current stage. Otherwise, trials will be conducted until at least one success has been found. Depending on the variation of the algorithm, it may begin this step as soon as the first success has been found, do all possible trials, or something in between. Whichever case it is, let $\mathbb{T}_i^{(m)}$ be the lowest or tied for the lowest trial utilization. Then, the next stage, $m+1$, begins, with $\forall j \neq i, U_j^{(m+1)} = U_j^{(m)}$ and $U_i^{(m+1)} = U'^{(m)}_i$.

Notice that $\mathbb{T}^{(m+1)} = \mathbb{T}_i^{(m)}$, which is the largest improvement found. This does mean that AFFPAA is greedy, taking the largest improvement it finds for each stage, that can be reached by improving one flow at a time. It will also never try an action that does not improve the existing state, and will try to shape each flow to the existing state as much as possible. While this is good for initial improvements, it remains to be seen if there are situations where this leads to finding a local minimum.

Another note is that, if i was the improvement taken for stage $m > 0$, then $\mathbb{T}_i^{(m+1)} = \mathbb{T}^{(m+1)}$, so it is always a failure. This comes from the fact L_O for i did not change from last stage.

5.4.1 Choice of Trials

AFFPAA, as stated above, does leave two key steps open: Choosing trials, and deciding to move to the next stage. This is because there are several ways to go about these steps, where which one is best may be situational.

The first approach is the slowest, but will always find the largest improvement: Try every $i \in \{1 \dots k\}$. This means the number of trials per stage is $O(k)$ (k in stage 0, $k-1$ in each subsequent stage). There can be a large benefit of picking the best success instead of a sub-optimal one. Additionally, this approach can use multi-threading very efficiently, as each trial can be performed simultaneously.

The second approach is the fastest, but has the highest risk of missing a useful move: conduct trials in an arbitrary order until a success is found. This can be modified for large k by waiting until a constant number of successes are found, or no more trials can be done. With a "first success" method, the average number of trials is $O(1)$ with respect to the number of flows, instead being roughly $\frac{1}{P}$ where P is the probability of a success. Obviously though, the first success may not be the best, and the risk of that increases with larger k.

The final approach partially tries each flow, then uses that initial result to decide on an order to conduct the trials. This approach is a middle of the road between the first two, being faster than approach one, and less risky than approach two. For each $i \in \{1 \dots k\}$, let the potential, P_i, be

$$P_i = \frac{\mathbb{V}\left(\mathbb{T}^{(m)}N - \sum_{j=1, j\neq i}^{k} L_{U_j^{(m)}}\right)}{G_i}$$

Note that the potential must be greater than or equal to one, as $\mathbb{T}^{(m)} = Ut(\sum_{i=1}^{k} U_i^{(m)}, N)$, and hence U_i is a flow on $\mathbb{T}^{(m)}N - \sum_{j=1, j\neq i}^{k} L_{U_j^{(m)}}$.

Notice that if the potential is 1, the trial will fail. So, if the potential is near 1, it's reasonable to assume any improvement would be slight, so conducting trials from highest potential to lowest is a sensible ordering. However, I cannot prove a direct relationship between potential and improvement size at this time, so it is possible it will not be the best. Which means that it may be good to sample the first few with the highest potential, and pick the best.

making an example of this is really hard, and it's not showing what I want it to, so it is blank for now.

5.4.2 Additional Tweaking

A major advantage of AFFPAA over linear programming for solving this kind of problem is that at each stage, the state of the system is a valid solution, better than what it started with. With a linear programming method, intermediate states have no guarantee to be solutions.

To increase the speed of trials, at the cost of potentially not finding the best improvement for that trial, one can partially complete the obstructed network problem. This is especially useful with the Modified Bisection method, as scaling up the flow for the lower bound, or scaling down the flow for the upper bound, will result in a flow with the goal value, but not optimal utilization. Since scaling a flow and checking utilization is $O(A)$, checking both bounds is reasonable, to be sure the best approximation is found. Due note though, if this shortcut is taken, the rule about the same flow not being an improvement twice in a row is no longer true, so it must be checked again.

On the other end, to be slower but more thorough, one can "branch" and try the second best option as well as the best, for two different next stages. Then, after some number of active stages are reached, the worst ones can be "pruned". There is a chance this may avoid local minimums, but as I have not proven one exists as of yet, this may be irrelevant.

The final thing to note is how k should impact how you use this algorithm. For large k, the potential stage to stage improvement tends to be lower, as each flow contributes less to the total load on the network. Meanwhile, if k is small, searching for a larger improvement is more worthwhile, as each flow has a huge part to play in the total load.

5.5 Questions remaining

There, unfortunately, were many things I was unable to accomplish or prove during my time with this problem. Each one of these things are open for more people to try and work on for the future.

1. **Does AFFPAA always terminate?**

   Since AFFPAA always improves each step, any situation where it does not terminate must be such that it makes smaller and smaller improvements forever. Currently, I do not have any idea what could possibly result in such a situation, but I cannot say one does not exist.

2. **If AFFPAA terminates, did it find the lowest utilization?**

   Again, I know that if it did find the lowest utilization, it would terminate. However, it is possible that there could be "local minimums", where one flow alone cannot be improved, but a combination changing at once would improve the utilization. Does always conducting every trial avoid local minimums? Are there situations where partial completion of a trial is preferable? Is the initial position of the maximum efficiency for each uflow a particularly good or bad state for avoiding local minimums? As should be clear, this is a very hard question to examine.

3. **Is there an efficient "smooth" approximation for the Affine Network problem?**

   One of the challenges with the Affine Network problem is that $V(X)$ is naturally "jagged", being made out of the minimum of affine functions. The methods I have found for this paper take advantage of computing points and first derivatives using existing max-flow calculation methods. However, I am left wondering if there's a way to efficiently compute a smooth approximation of $V(X)$. If there was, then higher level derivatives could be used on the smooth version, to earn more information about the structure as a whole.

4. **"Native" multi-flow techniques.**

   Algorithms for single flows on a network are well understood, but for multi-flow problems, the best methods seem to be getting rid of the "intuition" by converting everything into linear equations or working with one flow at a time to use single flow algorithms. A technique made for two or more flows " natively ", without using single-flows and then trying to make them work together, would be very helpful not just for this problem, but other multi-flow problems. However, I did not find much work in this area.

References

[1] J. A. Bondy, *Directed Graphs*, pp. 1–24. Elsevier Science Ltd/North-Holland, jun 1976.

[2] R. Diestel, *The Basics*, pp. 1–34. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017.

[3] S. Ruj, "Lecture 8: Paths, cycles and connectedness." 2014.

[4] J. A. Bondy, *Directed Graphs*, pp. 171–190. Elsevier Science Ltd/North-Holland, jun 1976.

[5] M. Fonoberova, "Optimal flows in dynamic networks and algorithms for their finding," in *Handbook of optimization in complex networks*, vol. 57 of *Springer Optim. Appl.*, pp. 363–403, Springer, New York, 2012.

[6] L. Schrijver, "Paths and flows: a historical survey," *CWI Quarterly*, vol. 6, no. 3, pp. 169–183, 1993.

[7] J. A. Bondy, *Directed Graphs*, pp. 191–211. Elsevier Science Ltd/North-Holland, jun 1976.

[8] C. P. LLC, "Algorithms and theory of computation handbook, "flow network"." dictionary entry, 1999.

[9] G. R. Waissi, "Worst case behavior of the dinic algorithm," *Applied mathematics letters*, vol. 4, no. 5, pp. 57–60, 1991.

[10] P. Hao, L. Hu, J. Jiang, and X. Che, "A stochastic adjustment strategy for coordination process in distributed networks," *COMPUTING AND INFORMATICS*, vol. 37, no. 5, 2018.

[11] J. Bang-Jensen and G. Gutin, *Flows in Networks*, pp. 95–170. London: Springer London, 2002.

[12] H. Karloff, *Linear Programming.* Modern Birkhäuser Classics, Boston, MA: Birkhäuser Boston, 1. ed., 1991.

[13] M. Cavers, S. Cioaba, S. Fallat, D. Gregory, W. Haemers, S. Kirkland, J. Mcdonald, and M. Tsatsomeros, "Skew-adjacency matrices of graphs," *Linear Algebra and its Applications*, vol. 436, pp. 3–4, 06 2012.

[14] S. Andrilli and D. Hecker, *Chapter 8 - Additional Applications*, pp. 513–605. Academic Press, 2016.

[15] I. Gutman and J.-Y. Shao, "The energy change of weighted graphs," *Linear Algebra Appl.*, vol. 435, no. 10, pp. 2425–2431, 2011.

[16] A. Schrijver, "On the history of the transportation and maximum flow problems," *Mathematical Programming*, vol. 91, pp. 437–445, Feb. 2002.

[17] F. Göring, "Short proof of Menger's theorem," *Discrete Math.*, vol. 219, no. 1-3, pp. 295–296, 2000.

[18] K. A. Berman, "Vulnerability of scheduled networks and a generalization of Menger's theorem," *Networks*, vol. 28, no. 3, pp. 125–134, 1996.

[19] D. Nace and M. Pioro, "Max-min fairness and its applications to routing and load-balancing in communication networks: a tutorial," *IEEE Communications Surveys Tutorials*, vol. 10, no. 4, pp. 5–17, 2008.

[20] H. S. Bin Obaid and T. B. Trafalis, "An approximation to max min fairness in multi commodity networks," *Comput. Manag. Sci.*, vol. 17, no. 1, pp. 65–77, 2020.

[21] N. Megiddo, "Optimal flows in networks with multiple sources and sinks," *Mathematical Programming*, vol. 7, no. 1, pp. 97–107, 1974.

[22] D. Nace, L. N. Doan, O. Klopfenstein, and A. Bashllari, "Max–min fairness in multi-commodity flows," *Computers & Operations Research*, vol. 35, pp. 557–573, Feb. 2006.

[23] P. Jensen, 1999.

[24] M. H. Akyüz, T. Öncan, and I. K. Altı nel, "Branch and bound algorithms for solving the multi-commodity capacitated multi-facility Weber problem," *Ann. Oper. Res.*, vol. 279, no. 1-2, pp. 1–42, 2019.

[25] H. Perfect, "Applications of Menger's graph theorem," *J. Math. Anal. Appl.*, vol. 22, pp. 96–111, 1968.

[26] G. A. Dirac, "Short proof of Menger's graph theorem," *Mathematika*, vol. 13, pp. 42–44, 1966.

[27] G. Chartrand, L. Lesniak, and P. Zhang, *Graphs & digraphs*. Textbooks in Mathematics, CRC Press, Boca Raton, FL, sixth ed., 2016.