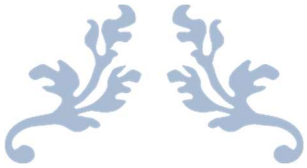


NETWORK OPTIMIZATION MODELS

A Deep Dive into the Network Simplex Method

Ethan Perry

Operations Research

Math 266

April 30th, 2025

Introduction

Network optimization models are a cornerstone of operations research, offering powerful tools to solve complex problems involving the flow of resources, information, or goods through interconnected systems. This paper delves into a variety of network problems, each addressing distinct challenges such as finding the shortest path, minimizing spanning trees, maximizing flow, and optimizing cost through networks. These models are not only theoretically robust but also highly applicable in real-world scenarios, from transportation and logistics to telecommunications and project management. By leveraging the structure of networks, these methods provide efficient solutions that often outperform general-purpose algorithms, making them indispensable in both academic and industrial settings.

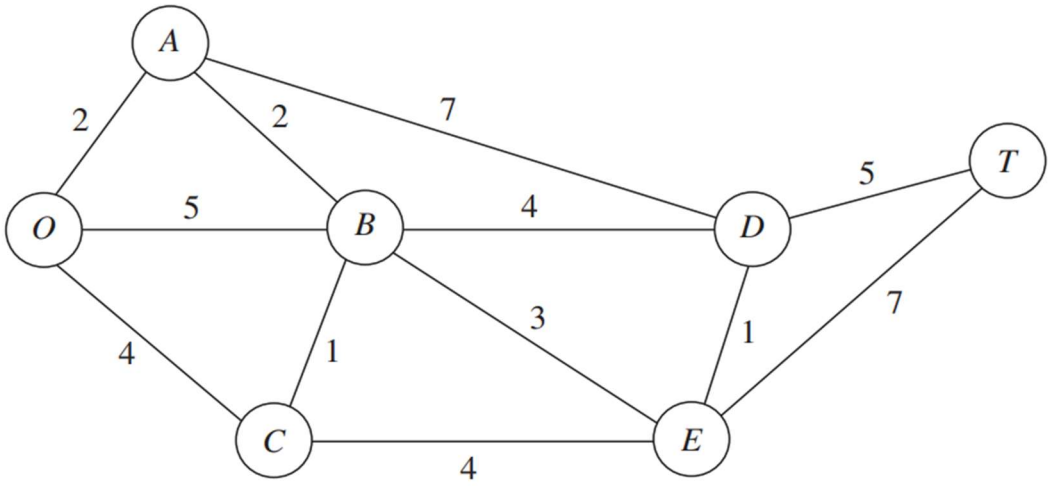
The chapter begins with a prototype example set in Seervada Park, illustrating practical applications of network optimization, such as determining the shortest tram route or laying telephone lines with minimal disruption. Subsequent sections explore fundamental terminology and methodologies, including directed and undirected networks, paths, cycles, and spanning trees. Key topics include the shortest-path problem, the minimum spanning tree problem, the maximum flow problem, and the network simplex method, each building on the previous to form a cohesive understanding of network optimization. Together, these topics provide a comprehensive toolkit for tackling a wide range.

Prototype and Terminology

The Seervada Park problem introduces a practical and intuitive prototype for applying the various methods described from this chapter. The Park needs to be strategic when it comes to sightseeing and hiking numbers to ensure preservation of the nature. The road system is shown below with straight line distances representing the windy and narrows park roads. Location O is the entrance to the park with roads leading around different points of interest, leading to Location T. Throughout many of the examples to come, the source or start will be O and the sink or destination will be T.

Various scenarios will arise in the park that can be solved by the various methods presented in this chapter. First, the park wants to create a special tram line that fast tracks to the site T since it is so popular. Management wants to find the shortest such path so that tourists just want to visit site T can do so efficiently.

A minimum path problem will be formulated and solved in order for the tram line to be carved through the network to satisfy the demand for site T. Second, the park wishes to install underground telecommunication wires to ensure all sites are connected for emergency communication lines. They wish to connect every site to the communication network while minimizing the amount of wire needed to be installed. A minimum spanning tree will be found through the park that equates to solving this telecommunication problem. Third, the park wishes to maximize the total number of visitors who can reach site T from a variety of path combinations. Finding the maximum flow will allow the park to keep as many trams as possible running constantly to maximize the amount of traffic to T. This equates to maximizing the flow through the network from the source point O to the destination T. Finally, the park would want to find the differing costs to send trams along various paths based on costs assigned to edges (not pictured below). There will then be designated paths that will cost tourists different amounts in order to travel across the network in specific ways. More detail will be added in order to formulate the minimum cost flow problem, and an entire simplex method will be drawn out in more detail, past this prototype example.



To understand Network optimization and the subsequent content, it will be vital to understand the terminology often used in this context. A **network** (also known as a graph), is a set of points, called **nodes**, connected by a set of lines, called edges or **arcs**. A set of nodes, also known as vertices, is often denoted by the letter V. Similarly, the set of all edges is denoted as the letter E. A network is the collection of V and E.

When talking about the connectedness of two nodes, a simple edge is called an **undirected arc**. There is no information about the direction of data traveling from one place to another, just that there is an established connection or **link**.

However, a **directed arc** contains information about a sender and a destination. In terms of network optimization, there is a supply node (the node supplying the data), which is sent along the directed arc to the demand node (the node receiving the data).

A network where all the edges in E are not directed is predictably called an undirected network. Undirected networks contain useful information regarding the connectedness of the graph but lacks directional information. Therefore, a network where the edges *are* directed is called a **directed network**.

In the more applied context of network optimization on a **directed network**, there are **supply nodes** (the node supplying the data), which are sent along the directed arc to the **demand nodes** (the node receiving the data). Any intermediate node is referred to as a **transshipment node** which acts like a middleman. The arc capacity of a directed edge is the allowable amount of data that can be sent alone that edge. In the context of the prototype example, imagining each road can only have a certain number of cars at a time, which would define its arc capacity.

A **path** is a set of unique (non-repeated) edges that connect two nodes in a network. For example, in Figure 1, a valid path connecting O to T would be OA – AD – DT. An invalid path would be OA – AD – DB – DC – CD – DT, since you D was hit twice. This idea of a path generalizes undirected and directed given the nature of the network being considered. A **cycle** is a directed path through a network that starts and ends at the same node. If O → B, B → C, and C → O, then that would form a cycle.

Shortest Path

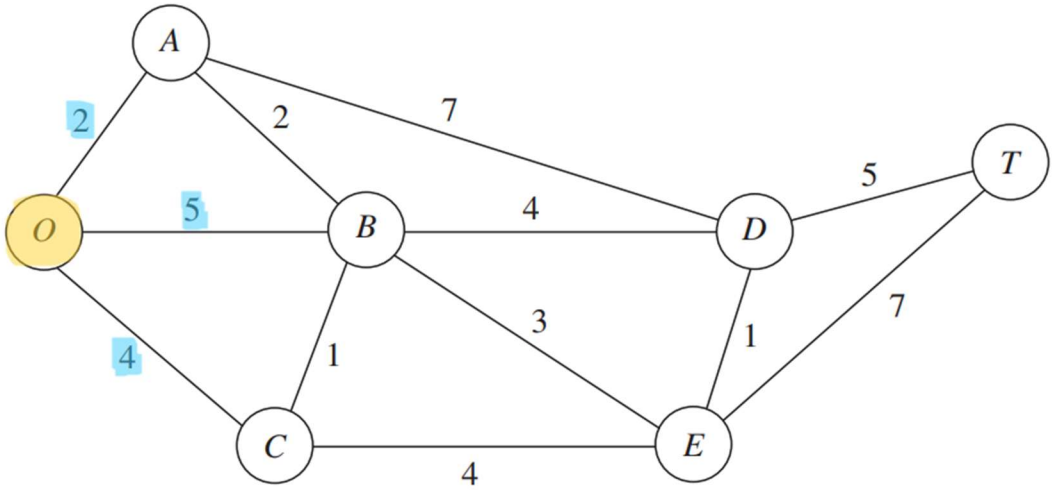
The shortest path problem is extremely common in graph theory and computer science applications. Finding the shortest viable path the links a given start and end node has been researched ad nauseum. It is essential to consider two different types of networks, weighted and unweighted. The essence of the problem is that considering all possible paths to the destination node, what is the optimal path that minimizes the total distance traveled.

In an unweighted graph, the solution follows a Breadth-first Search (BFS) approach. BFS is a fundamental graph traversal algorithm that explores graph level by level, starting from a designated source node. It systematically visits all nodes at a given distance from the source before moving on to nodes at the next level, making it especially useful for finding the shortest path in unweighted graphs. BFS uses a queue data structure to keep track of nodes to visit, ensuring

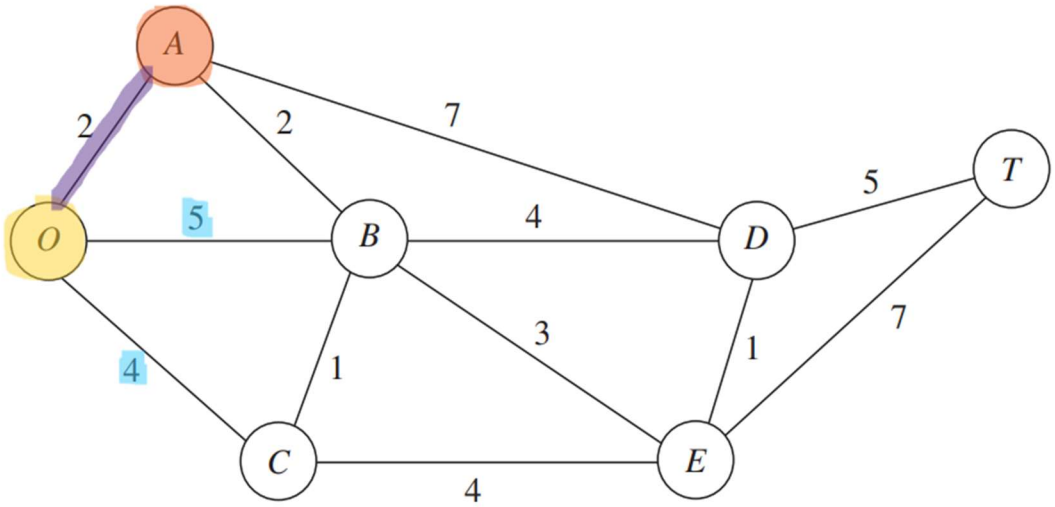
that nodes are explored in order they are discovered. Its predictable traversal pattern and simplicity make it a key building block in many network algorithms, including those used in maximum flow and minimum cost computations.

However, in a weighted graph, a simple BFS solution is not robust enough due to the idea of local and global optimization. The local optimization solution would be to choose the shortest edge connected to the current node you are considering. However, it is possible to lose sight of the global optimal solution when doing simple BFS. There is a common algorithm that considers a minimum path through directed networks called Dijktra's Algorithm. This algorithm falls outside of the scope of this paper but can be researched by the reader.

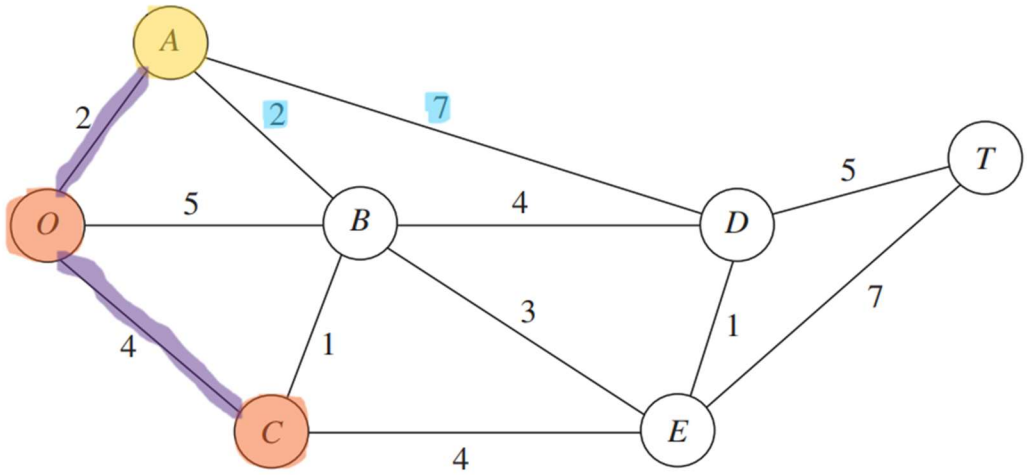
We will now consider the algorithm laid out by the text and then implemented by excel solver. You start by finding the nth nearest node to the origin at each iteration until that node is equal to the destination. This means for n=1, you consider all the paths from the origin to a node where you only travel along 1 arc. For n=k, you consider all the paths from the origin to nodes where the path taken consisted of k number of arcs traversed. Once you reach the destination, you backtrack through the minimum connected distances from previous iterations to construct the shortest path. On the next page, the algorithm will be demonstrated on the prototype problem for the Seervada park.



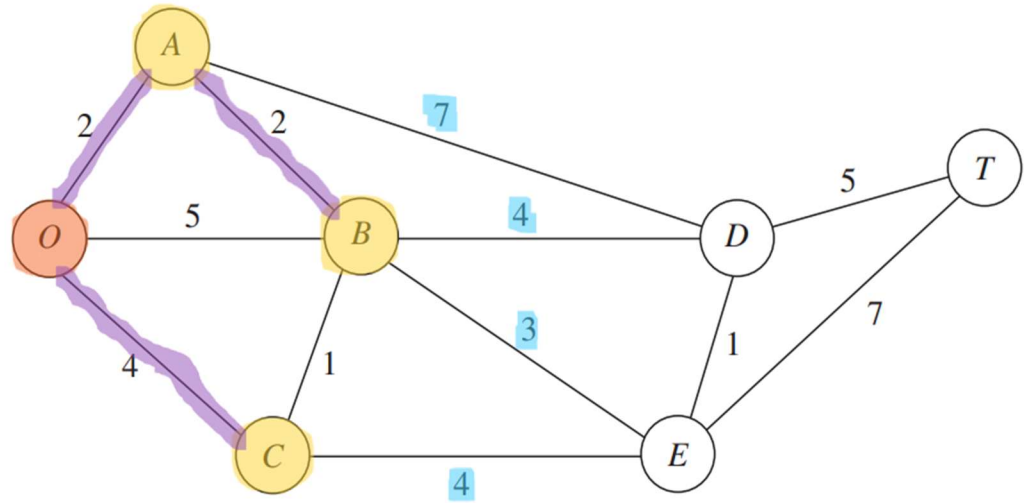
n	Solved Node Connected to Unsolved	Closest Connected Unsolved Node	Total Distance	*N*th Nearest Node	Min Distance	Last Connection
1	O	A	2	A	2	OA



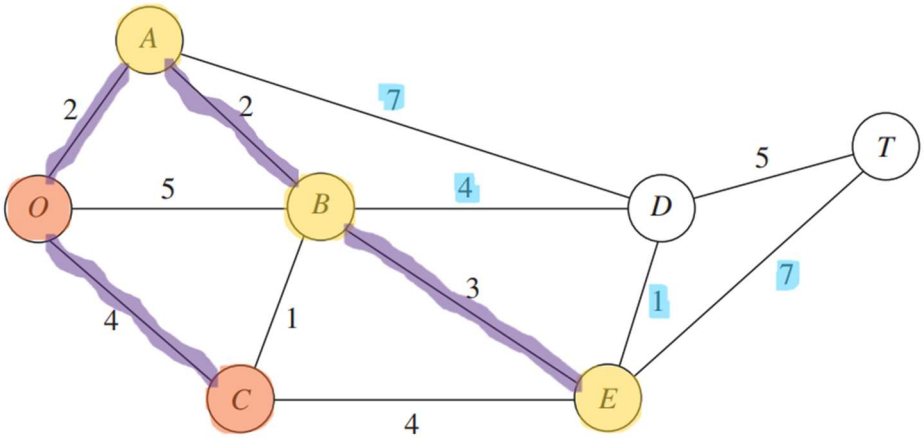
n	Solved Node Connected to Unsolved	Closest Connected Unsolved Node	Total Distance	*N*th Nearest Node	Min Distance	Last Connection
2	O	C	4	C	4	OC



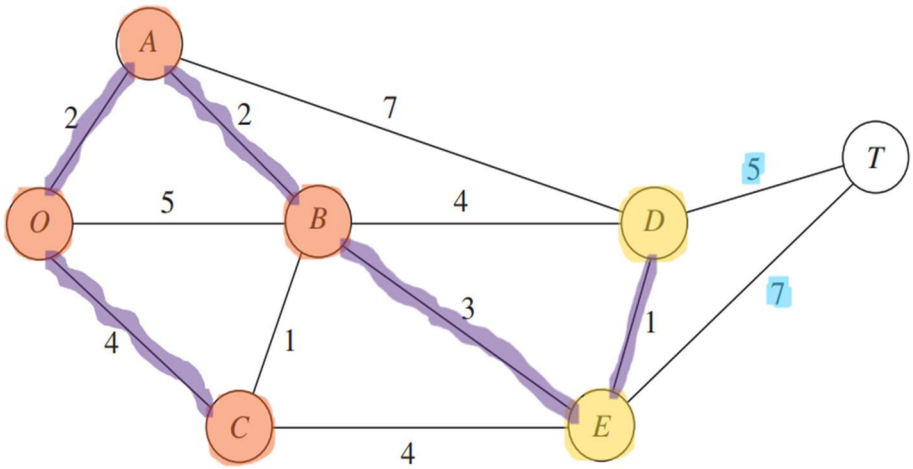
n	Solved Node Connected to Unsolved	Closest Connected Unsolved Node	Total Distance	*N*th Nearest Node	Min Distance	Last Connection
3	A	B	2+2=4	B	4	AB



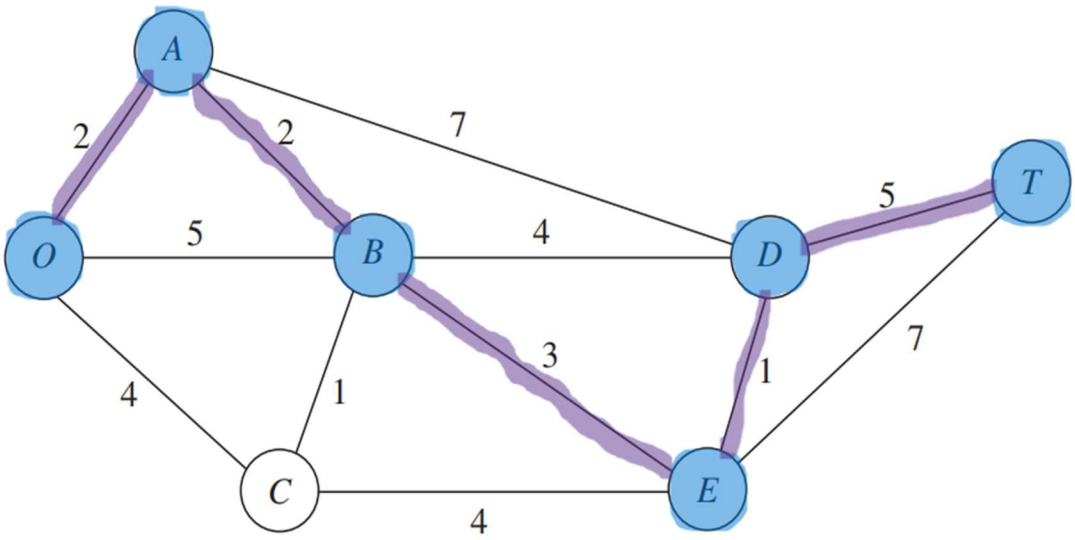
n	Solved Node Connected to Unsolved	Closest Connected Unsolved Node	Total Distance	*N*th Nearest Node	Min Distance	Last Connection
4	A **B** C	D **E** E	2+7=9 **4+3=7** 4+4=8	**E**	**7**	**BE**



n	Solved Node Connected to Unsolved	Closest Connected Unsolved Node	Total Distance	*N*th Nearest Node	Min Distance	Last Connection
5	A B **E**	D D **D**	2+7=9 4+4=8 **7+1=8**	D **D**	8 **8**	BD **ED**



n	Solved Node Connected to Unsolved	Closest Connected Unsolved Node	Total Distance	*N*th Nearest Node	Min Distance	Last Connection
6	**D** E	**T** T	**8+5=13** 7+7=14	**T**	**13**	**DT**



OA	→	AB	→	BE	→	ED	→	DT	Sum
2	+	2	+	3	+	1	+	5	= 13

So, from this algorithm, we have found the global minimum for this network. This solution found by this process is the purple path above with a minimum cost of travel through the network of 13 units. However, for networks any larger than this example problem, a computational approach will be taken. On the next page, the excel set up for the min path problem will be laid out.

Here is the set up for excel solver. We send one unit of flow through the system to minimize the cost. The cost is the sum product of the active arcs in the route (with a value of 1) multiplied by their distances. We see that the path given by excel matches the algorithm done above. See from the excel solver output below that the resulting shortest path solutions are the same.

Seervada Shortest-Path Solution

From	To	Route	Distance
O	A	1	2
O	B	0	5
O	C	0	4
A	B	1	2
A	D	0	7
B	C	0	1
B	D	1	4
B	E	0	3
C	B	0	1
C	E	0	4
D	E	0	1
D	T	1	5
E	D	0	1
E	T	0	7

Distance	13	=SUMPRODUCT(Route, Distance)

Nodes	Net Flow		Supply/Demand	
O	1	=	1	=SUMIF(From, O, Route) - SUMIF(To, O, Route)
A	0	=	0	=SUMIF(From, A, Route) - SUMIF(To, A, Route)
B	0	=	0	=SUMIF(From, B, Route) - SUMIF(To, B, Route)
C	0	=	0	=SUMIF(From, C, Route) - SUMIF(To, C, Route)
D	0	=	0	=SUMIF(From, D, Route) - SUMIF(To, D, Route)
E	0	=	0	=SUMIF(From, E, Route) - SUMIF(To, E, Route)
T	-1	=	-1	=SUMIF(From, T, Route) - SUMIF(To, T, Route)

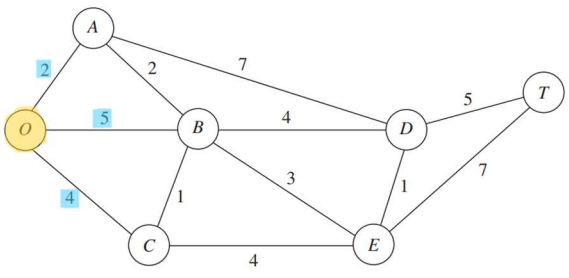
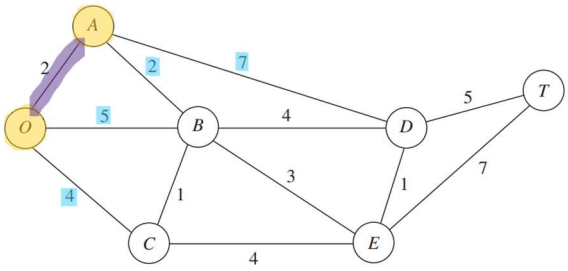
Minimum Spanning Tree

A spanning tree is a set of edges that connect each node in a graph without creating cycles. A minimum spanning tree is the tree that minimizes this connectedness. The concept and algorithm are generally straight forward through example.

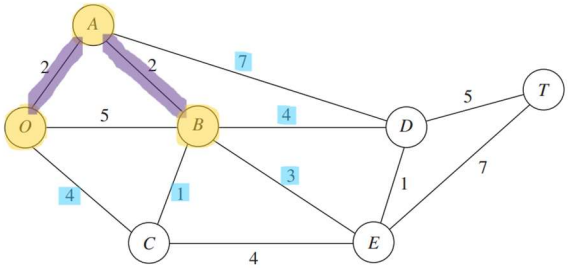
The general algorithm is as follows.

1. All nodes start disconnecting from each other
2. Choose a node arbitrarily (Node O) and connect it to its closest (min distance) neighbor (node A)
3. Identify the closest unconnected node to any of the connected nodes (closest neighbor to either O or A)
4. Repeat this process until all nodes are distinctly connected

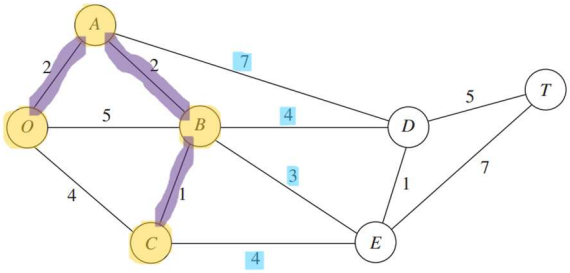
In the illustration below, yellow nodes are ones that are solved and connected into the spanning tree. The blue highlighted edges represent the candidate weights for the next step in connection with the graph. The algorithm chooses the minimum of these blue candidates. Continue this pattern of solving nodes and considering candidates until no more nodes are disconnected.

Iteration 1: Connect O to closest neighbor A	Iteration 2: Connect A to B
	

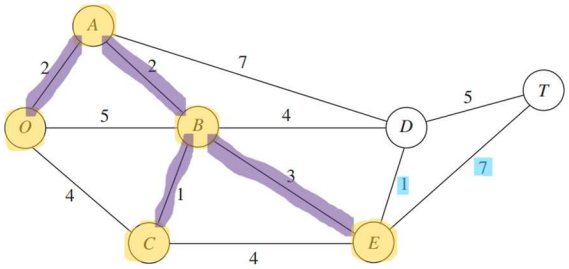
Iteration 3: Connect B to C



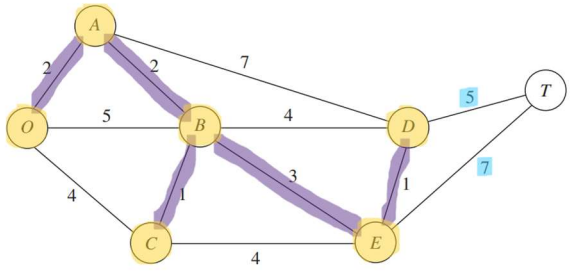
Iteration 4: Connect B to E

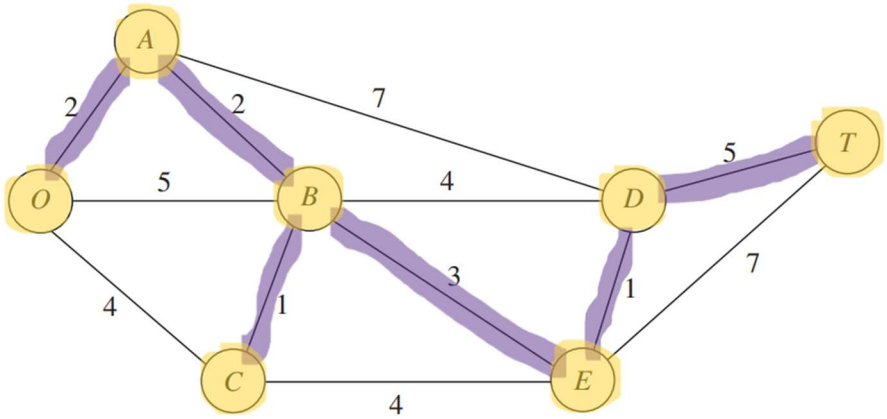


Iteration 5: Connect E to D



Iteration 6: Connect D to T



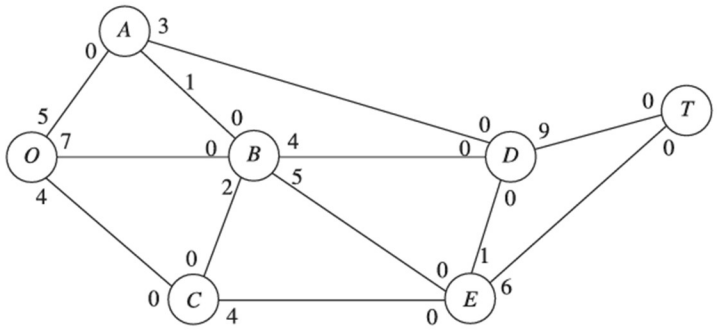


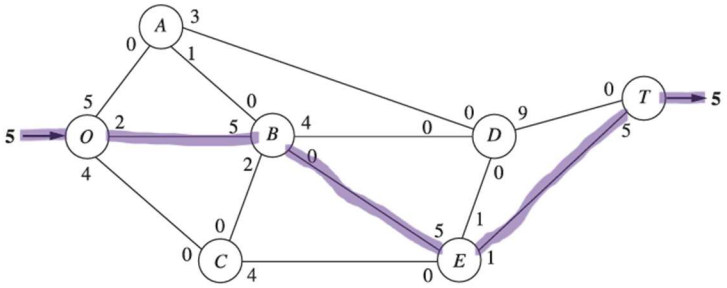
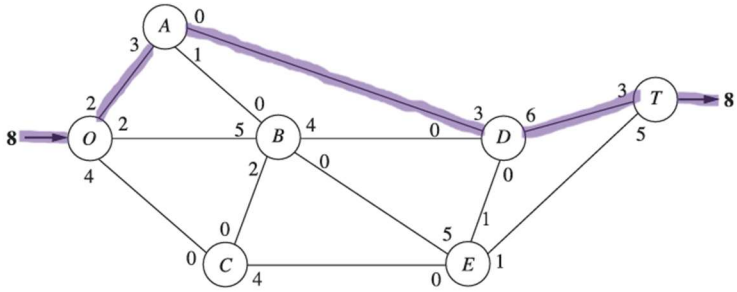
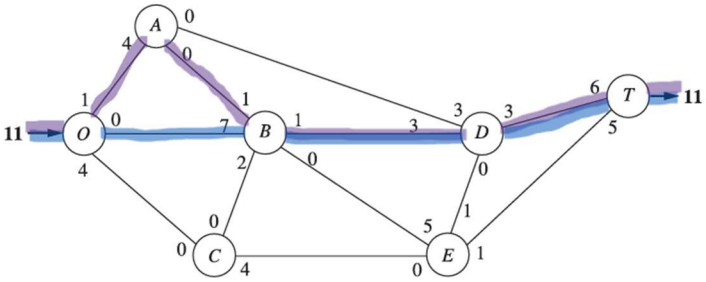
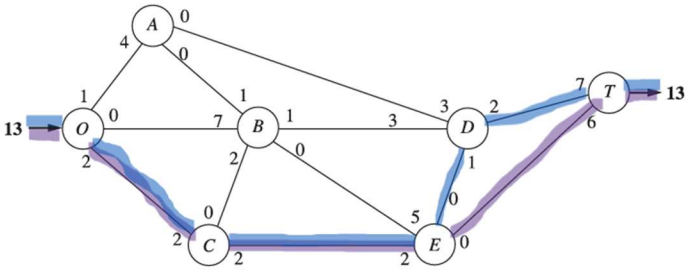
We now see that each node is connected to the network without any cycle appearing. Also note that the sum of the connecting edges is the minimum such configuration. In the case of the park, these are the routes along which the minimum amount of wire must be laid in order to fully connect the park's communication systems.

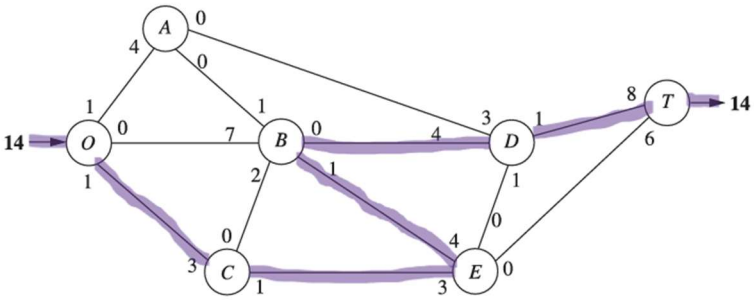
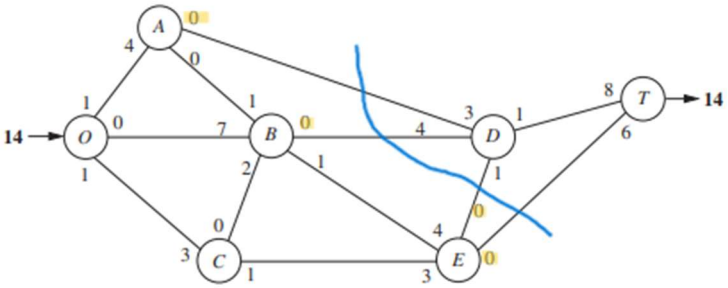
Maximum Flow Problem

The maximum flow problem stands as a fundamental problem type in the realm of network optimization due to its wide range of applications it supports. From the allocation of resources in supply chains to the routing of data in telecommunications networks, the ability to determine the most efficient flow through a network under given constraints is vital. The core premise is maximizing the amount of flow from a designated source to a designated sink. The applications of this powerful class of problems recur across engineering, economics, logistics, and computer science. This section builds on the theoretical foundation of the max flow model, expanding it to accommodate various practical extensions that reinforce its real-world applicability.

In this section, we'll explore advanced algorithmic techniques, such as capacity scaling, integral flows, minimum cuts, and strong duality relationships. These developments underscore the significance of this problem class, not merely as an academic exercise, but as a toolkit for modeling and solving complex systems. Ultimately, this aims to improve the readers' understanding of flow-based optimization by introducing them to the breadth of scenarios in which the max flow problem and its variants can be applied.

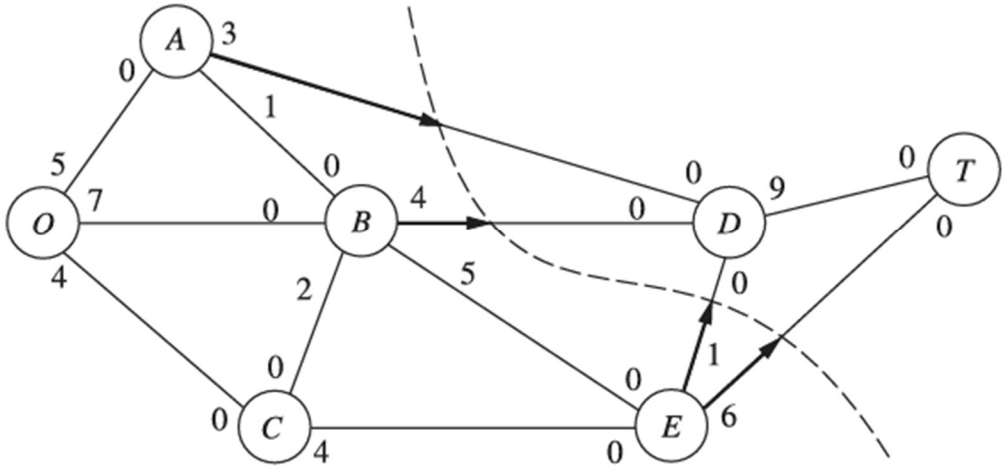
This is the initial network diagram. The numbers coming out of a node demonstrate the arc supply. Take the arc OA for example. OA can send 5 units across this arc and the corresponding 0 and node A indicates that nothing has been sent through A along this arc yet.	

Iteration 1 begins with sending 5 units through the system. Traced in purple is the path along which these 5 units travel, O → B → E → T. Notice the residual capacities along this line. Before, the arc OB had a capacity of 7, but now it has a capacity of 7-5=2.	
Iteration 2 starts with the network after the first iteration, so now there are 8-5=3 units being sent through this path O → A → D → T. Not the updated residual network and capacities.	
Iterations 3 and 4 are drawn in the same diagram. See that one unit is sent along the purple path O → A → B → D → T, and that two units are sent along the blue path O → B → D → T. This results in a total network flow of 8+1+2=11 thus far.	
Iterations 5 and 6 are also shown in the same diagram. Quiz for the reader: see if you can derive how many units are being sent across each highlighted path. One unit is being sent across the blue path O → C → E → D → T and one unit is being sent along the purple path O → C → E → T.	

Finally, the 7th iteration is also the final iteration after one unit is sent along O → C → E → B → D → T. We have now sent as many units through the network as possible and we stop the algorithms process.	
Notice this is the final solution since there is no way to ship more units into D. Every entering arc into D has an arc allowance of 0. This means that you could start to ship along other feasible arcs, but none of those units will reach T due to the D bottleneck.	

The max-flow min-cut theorem is a fundamental result in network optimization that establishes a deep connection between the maximum flow in a network and the concept of minimum cuts. A cut in a network is a partition of the nodes into two disjoint sub graphs, separating the source from the sink, and its capacity is the sum of the capacities of all arcs crossing the cut in the direction from the source to the sink. The theorem states that the maximum flow from the source to the sink is equal to the minimum capacity across all possible cuts in the network. This method not only provides a powerful tool for verifying optimality in maximum flow problems but also offers insights into the bottlenecks and critical pathways that constrain flow within the network.

In the min cut below, we see that the sum of arcs being cut are 3+4+1+6=14. Note that 14 is the optimal solution from the method shown above. Note that this method requires cutting all possible cuts in the network and for the problem of this size of larger, it is not feasible to find the optimal this way. However, taking an arbitrary cut gives you a firm upper bound for the feasible optimal solution.



From	To	Flow		Capacity
O	A	3	<=	5
O	B	7	<=	7
O	C	4	<=	4
A	B	0	<=	1
A	D	3	<=	3
B	C	0	<=	2
B	D	4	<=	4
B	E	3	<=	5
C	E	4	<=	4
D	T	8	<=	9
E	D	1	<=	1
E	T	6	<=	6

Max Flow	14	=	C21

Nodes	Net Flow		Supply/Demand
O	14		
A	0	=	0
B	0	=	0
C	0	=	0
D	0	=	0
E	0	=	0
T	-14		

=SUMIF(From, O, Flow) - SUMIF(To, O, Flow)
=SUMIF(From, A, Flow) - SUMIF(To, A, Flow)
=SUMIF(From, B, Flow) - SUMIF(To, B, Flow)
=SUMIF(From, C, Flow) - SUMIF(To, C, Flow)
=SUMIF(From, D, Flow) - SUMIF(To, D, Flow)
=SUMIF(From, E, Flow) - SUMIF(To, E, Flow)
=SUMIF(From, T, Flow) - SUMIF(To, T, Flow)

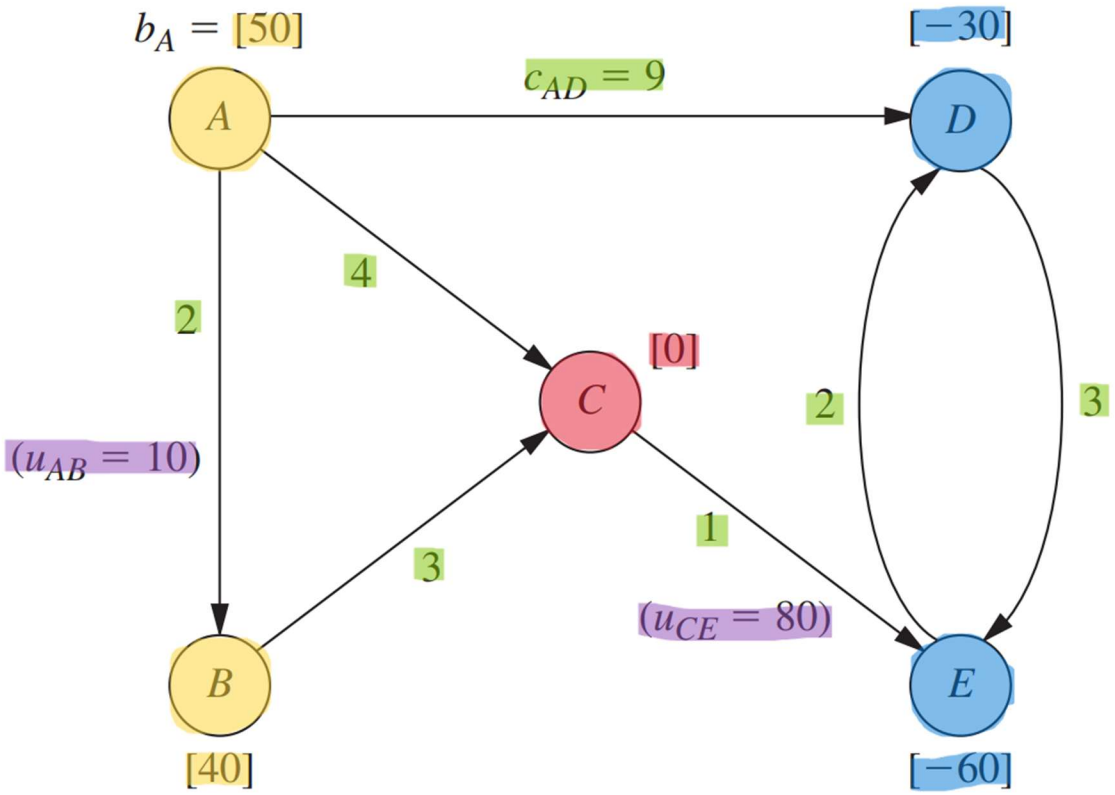
Minimum Cost Flow

Consider a directed and connected network with n nodes and at least one supply node and one demand node. With this definition, we can establish a minimum cost network flow in which supply = demand and the cost to ship all units through the network is minimized. Below defines the necessary variables to formulate this optimization model.

- x_{ij} - the flow through the arc i → j
- c_{ij} - the cost per unit flow through the arc i → j
- u_{ij} - the arc capacity (max flow) through the arc i → j
- b_i - the net flow generated at node i
  - $b_i > 0$ - if node i is a supply node
  - $b_i < 0$ - if node i is a demand node
  - $b_i = 0$ - if node i is a transshipment node

Objective: Minimize $W = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$
Subject to:
$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = b_i$ for each node i
and $0 \leq x_{ij} \leq u_{ij}$ for each arc i → j

In other words, get the total cost for each arc i → j which is the product of the amount of units being shipped across that arc and the cost per one unit. The constraints state that the total flow out of node i to j subtracted by the total flow into node i from j is equal to b_i and that the total flow across any arc must not exceed its arc capacity constraint. Please see the illustration below to see application of this nomenclature.



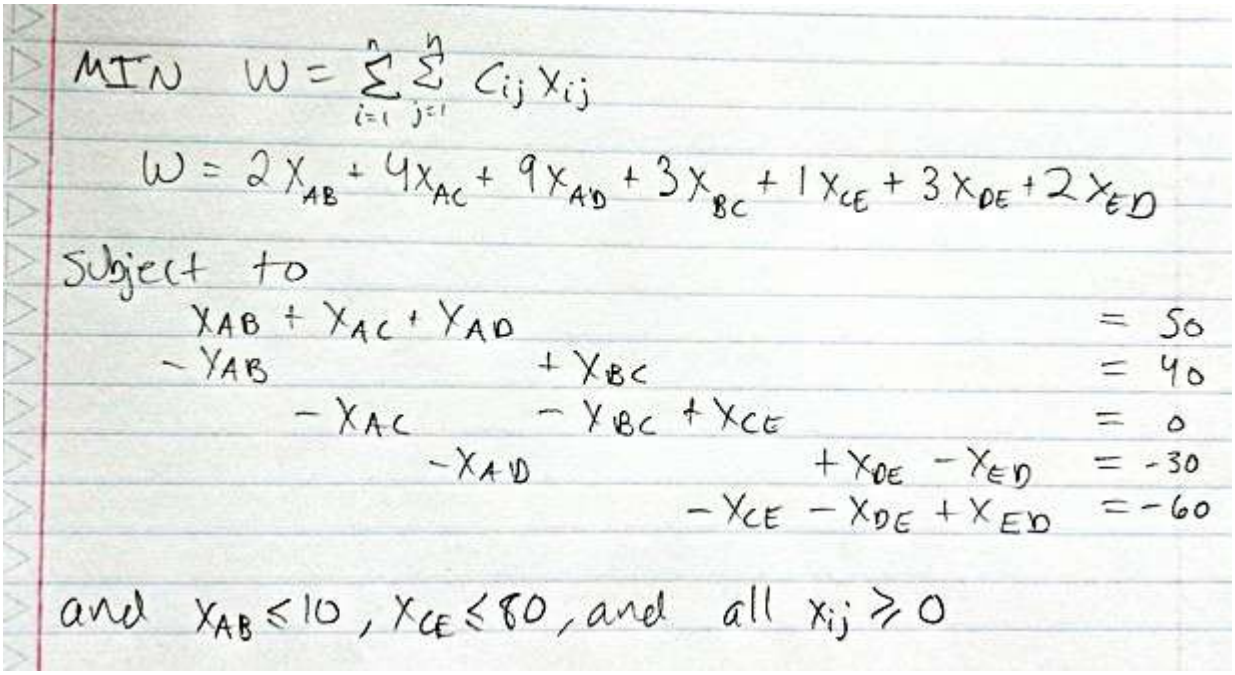
In yellow are the supply nodes because they have a positive net flow value. Node A can ship 50 units throughout the network. The red node, C, is the transshipment node since it has a net flow of 0. The magnitude of units received by a transshipment node will always equal the magnitude shipped away from it. And in blue, the demand nodes, characterized by their negative net flows. More is being received than sent out.

Highlighted in green are the costs for each arc. $C_{AD} = 9$ meaning that one unit shipped from A → D costs 9 units of cost. And finally in purple are the arc capacities. A → B can only send 10 units at most, and arc C → E can send 80 units at most. Note that any unconstrained arc could theoretically be any positive real number, so long as the entire net balance of the network is 0.

Notice that in network flow optimization models, such as the minimum cost flow problem, the set up includes a flow conservation constraint at each node to ensure that inflow equals outflow, adjusted by any supply or demand. However, because the total supply must equal total demand for a feasible flow to exist, these constraints are not all independent. Specifically, only *n-1* of the *n* flow constraint equations are linearly independent; the remaining constraint can be derived from the others. This reflects the underlying structure of the network, where the flow

values possess *n-1* degrees of freedom. Including all n constraints can therefore lead to redundancy and, in some cases, degeneracy in the simplex solution space, where multiple basic feasible solutions correspond to the same vertex in the feasible region. To avoid this and simplify the model, it is standard practice to omit one of the constraints, without loss of generality or correctness.

So, what would a feasible solution look like? The basis (which is a vector of primal flow values) would satisfy all the constraints below. Picking a starting basis can be tricky however, since the net flow across the entire network must balance. In the section regarding the network simplex method, algorithms used to establish a basis feasible vector will be detailed.



Here is the result of the min cost flow network problem after using Excel solver. We see that the flow through the system is net zero since all of the supply and demands for a respective node match their net flow constraint value. Also, the arc capacity constraints are met (highlighted in grey). This means that the minimum cost to ship units across and through this network is 490. We will see later that a problem given in context will illustrate the importance of this type of problem.

From	To	Flow	Capacity	Cost
A	C	40		4
A	D	10		9
A	B	0	10	2
B	C	40		3
C	E	80	80	1
D	E	0		3
E	D	20		2

Nodes	Supply/Demand		Net Flow
A	50	=	50
B	40	=	40
C	0	=	0
D	-30	=	-30
E	-60	=	-60

Total Cost	490

=SUMPRODUCT(Flow, Cost)

=SUMIF(From, A, Flow) - SUMIF(To, A, Flow)
=SUMIF(From, B, Flow) - SUMIF(To, B, Flow)
=SUMIF(From, C, Flow) - SUMIF(To, C, Flow)
=SUMIF(From, D, Flow) - SUMIF(To, D, Flow)
=SUMIF(From, E, Flow) - SUMIF(To, E, Flow)

Generalizations of Min Cost Flow

The min cost flow problem is the pinnacle of network optimization models because of its robustness and ability to generalize for many types of network inputs. This is the reason why an entire simplex method was developed and why there is so much research in the application of network optimization.

Above, the set up for an optimization model was defined on a very nice and average network for min cost flow. However, there are several special cases where some slight modifications can allow for the network simplex method to be applied on related problems.

The Transportation Problem developed in the previous chapter does have its own method for solving it, but it can be modified to fit the network model above. To augment this problem, a supply node provides flow for each source and a demand node is receiving flow from each destination. In this way, there are no transshipment nodes, and the flow from supply to demand equates to the flow across a generic arc $i \rightarrow j$ corresponding to source and destination. This structure allows the transportation problem to be represented as a special case of the minimum cost flow problem, where all arcs are directed from supply nodes to demand nodes, and capacities are typically infinite or sufficiently large to accommodate all feasible shipments. By mapping supply and demand directly to node balances in a network, we gain the flexibility to apply network-based algorithms, such as the network simplex method, while preserving the original cost-minimization objective of the transportation model.

The Assignment Problem is solvable when supply and demand are equal, and each source gets sent to uniquely one destination. To formulate it as a minimum cost flow problem, the number of supply and demand nodes must be equal, and the net flow at all supply nodes must be 1, whereas the net flow at each demand node is -1. This means each supply wants to send out exactly 1 unit of flow, and since there are equal number of demand nodes, each supply will map to exactly one demand node, thereby neutralizing the network. The arcs picked to be the "assignment vectors" (if you will) and the cost can easily be minimized with the network simplex method.

The Transshipment Problem case is actually very close to the min cost flow problem, except that there are no arc capacity constraints in this problem type. The Transshipment Problem is frequently used as a generalization of the transportation problem, where there can be transshipment nodes between source and destination. It turns out that many shipping problems in application have

these intermediate points, so the minimum cost flow setup is very applicable with not much augmentation at all.

The Shortest Path Problem discussed earlier in this chapter is very common in graph theory and computer science algorithms. There are well defined search algorithms across weighted networks, namely Dijkstra's algorithm. However, it involves several advanced data structures and some algorithmic thinking that is a big specialized. Since the shortest path is typically done with an undirected graph, each arc $i \rightarrow j$ will get an accompanying arc $j \rightarrow i$. The weight of the complementary arc will just be the same weight (typically distance) as the original arc. Then, the given supply node will have a supply of one unit and find the shortest path (minimum cost) to get its one unit to the demand node with demand one. Every intermediate node will have a net supply of 0 and the path traced will be the minimum cost route to traverse the network.

The Maximum Flow Problem, with three main adjustments, falls well into the formulation for the min cost flow problem. Firstly, you set all arc costs to 0 since the goal of this problem is to find the maximum units that are able to be shipped across the network, not necessarily the cost. Next, find value F, which is a safe upper bound for the maximum amount of flow that can be sent through this network. Some sort of min cut approximation can be done, but the goal is to choose an upper bound that is comfortably large so that the max flow can be sent, and excess can be ignored. Send F through the supply and have the destination have a demand of F. Thirdly, create an arc directly from source to sink with a cost of big M. In this way, all flow will be sent through the network because the arcs have no cost. Once it is not possible to send any more flow through arcs with filled capacity, ship the remaining units from F along the source to sink arc. The difference in F and what was shipped across the big M arc will be the max flow for the network.

I hope it is apparent how powerful the minimum cost flow problem becomes, and why there will be such emphasis placed on understanding the network simplex method for this problem type. So many other optimization problems seen in previous sections with their own unique methods can all be generalized as one standard method after some augmentation of the original problem.

Network Simplex Method

When setting up a simplex, we first must start with a basic feasible solution. In linear programming, we start at the origin and then work through the simplex method, traveling to corner points and assessing their optimality. So, what would be the equivalent in the network application? What does a basic feasible solution look like?

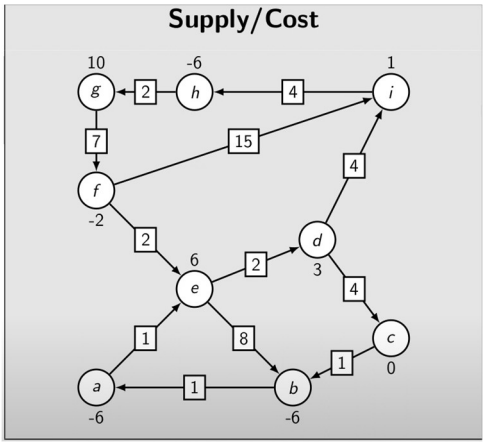
The **fundamental theorem for the network simplex method** says that basic solutions are *spanning tree solutions* (and conversely) and that BF solutions are solutions for *feasible spanning trees* (and conversely).

This implies that a spanning tree solution is a basis. Whether that basis is feasible or not is to be discussed, but an MST is automatically a basis. For a spanning tree solution to be feasible, you must find a set of flow weights that result in a totally balanced tree. This means that the net flow over the network is 0. To achieve this, all of the edges in that spanning tree are basic $x_{ij} > 0$ and all the arcs not included in this solution are non-basic, $x_{ij} = 0$. It turns out that these non-basic arcs have dual slack values that correspond to reduced cost when they enter, similar to the LP simplex method. When all of the dual slack variables are also non-negative, implying the overall cost cannot be reduced further, this indicates the optimal. We will later discover this is guaranteed by strong duality of solving both the primal and dual problem simultaneously.

Since our objective function is to minimize cost across a given flow network, we want to visualize this by transforming the supply/cost network to a flow network. We wish to maximize the flow around the network while keeping in mind the cost weight of each basic arc. Ultimately, we will have a diagram that shows the primal flow variables, the dual optimization variables (related to cost), and the dual slack variables (related to reduced cost). The definitions of all three will be built out in the following sections.

We will begin with this supply/cost diagram showing the shipment cost for units around this connected and directed network. For the network simplex method to work, we assume a root node which typically doubles as the supply node. However, note that a valid network in the simplex method may not have a clearly defined source and sink nodes, but can be easily modified to accommodate that. For example, with this network, we treat node a as the root node. This allows us to have a clearly defined node when we are talking about the minimum

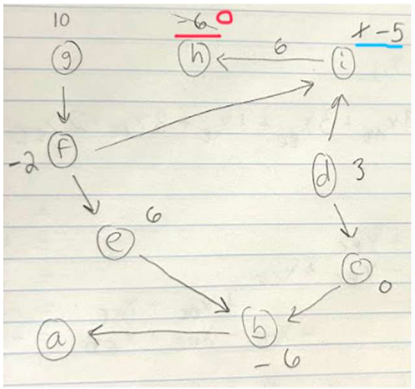
spanning tree. Since trees must have a root by definition, this will become handy in the following procedure.



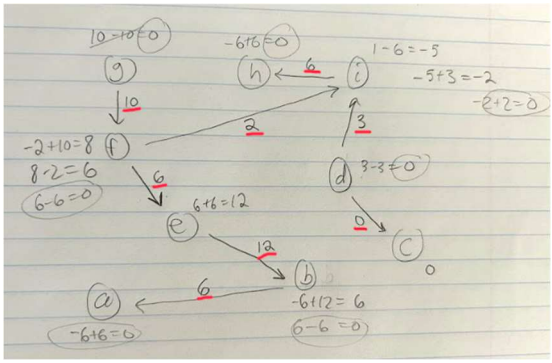
There are designated algorithms to generate a valid basis on a network. We know that it must be a spanning tree, but there are complex and optimized methods to achieve this. For the purpose of this paper, we will assume that generating this initial basis is taken care of by one of these algorithms. For the curious reader, the algorithms rely on graph traversal algorithms such as breadth first search or depth first search.

Primal flow variables

We wish to find the flow in the network that will result in overall balance at the root node a. To achieve this, we start at the terminal 'leaf' nodes and ship all of their supply through the basis to the root. Note that shipping supply in the opposite direction of an arc is the same as sending a negative amount. Better yet, think of this as the demand node saying, "we needed *x* less units to be balanced". Below, see the first step and final step, assuming the reader can follow the algorithm to fill in the gaps as an exercise.



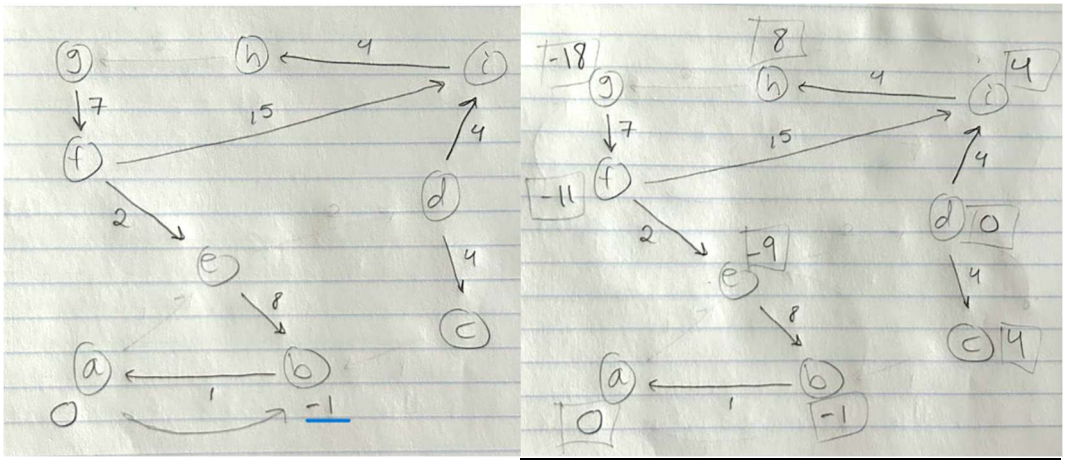
The process begins at the leaf node h which has a supply of -6 as the start. We send -6 backwards along the edge 'h-i' so that the primal flow value is -(-6) = 6. This results in node i has to send 6 units to satisfy that new arc (which has flow 6). This results in node i now having a supply of -5 == a demand of -5. We now ask the reader to continue working up the tree toward the root node, assigning primal flow values and updating the supply at each node until you reach the root with supply 0.



Some work is shown below, tracing the steps in the process for the reader's aid. The numbers underlined in red are the primal flow variables. This means that the basis composed of these arcs is primal feasible since the flow values are all non-negative. Since this is a basis, we have found a feasible solution to the network problem. The optimal value would be the flow values (underlined in red) times the cost to ship across each respective edge.

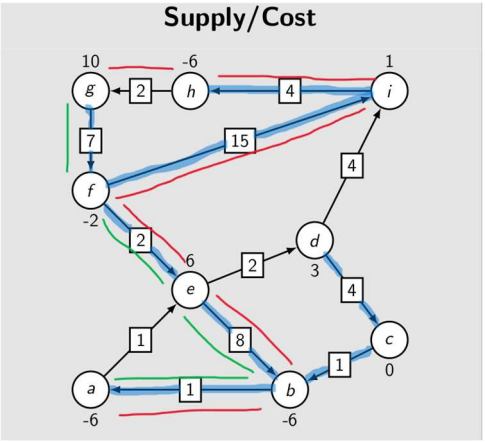
Dual optimization variables

Since we know a dual optimization variable is related to the constraints in the primal problem, these values should relate to the cost of flow in from the supply/cost diagram. It turns out that a dual optimization variable is the shadow price for that node. If you send one more unit along the network, the optimization value increases by that unit amount. More formally, the dual optimization variable at a node i is the price to ship one unit from the root, along the basis, to that node. Similar to as above, I will lay out the algorithm, then trace the first step, leaving the exercise to the reader. Our objective is to start with the root node and ship a unit along our given basis to find the price.



Dual Slack

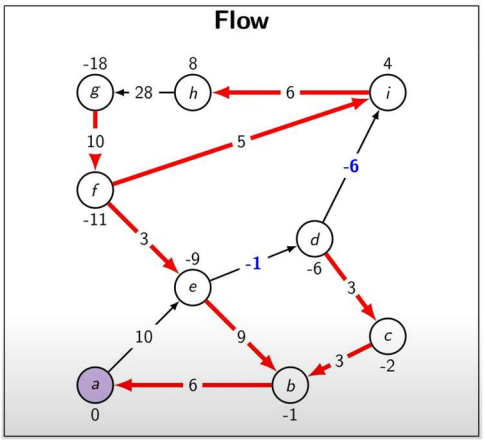
Recall the definitions of $x_{ij}, c_{ij}, u_{ij}, b_i$ which correspond to the flow, cost, capacity and net flow at a given arc i → j. We will add a definition for the dual slack of a non-basic arc so that the primal network simplex method becomes clearer. The dual slack is defined as $S_{ij} = p_{ij} - p_{bj}$. where p_{ij} is defined as the cost to send one unit along the basic arcs to i and then to j and p_{bj} is defined as the cost to send one unit along the basic arcs straight to j. Let us use our prototype example for this section to illustrate this.



Here is a given supply and cost diagram of a network. In blue, is the initial basic feasible solution. Later, the method to generate an initial BFS will be discussed. p_{hg} is denoted in the red line and p_{bg} is the green line. See that the sum traveling along the red line is p_{hg}= -1 + -8 + -2 + 15 + 4 + 2 = 10 and p_{bg} = -1 + -8 + -2 + -7 = -18. So the dual slack for the non-basic arc h → g is 10 - - 18 = 28. This means that if the arc h → g was to enter the

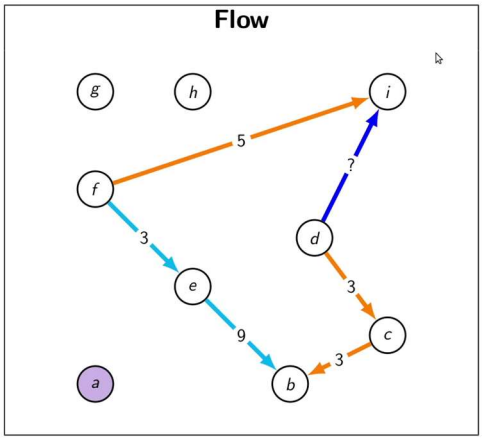
basis, the total cost to the objective function would be a 28 unit increase. This implies that similar to the LP simplex method, we will choose non-basic elements that have negative coefficients, since they will result in a minimization to our objective function.

Flow Diagram



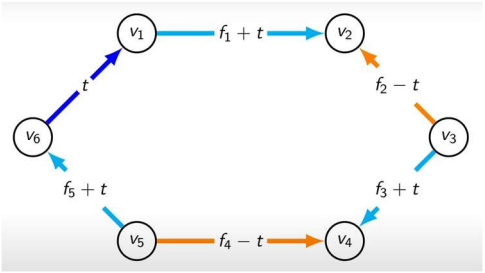
So here is our final picture. We have the primal basis which is the red arcs defining a spanning tree of the network. It turns out that this basis is also primal feasible since all of the arcs in the basis are non-negative. However, we know that our solution is not optimal because of the -6 on the arc d → i and the -1 on the arc e → d. To interpret, this means that if the arc d → i enters, our optimal solution will decrease by a unit cost of 6. We will next explore how this arc can enter the basis, what a primal network pivoting strategy looks like, and how we know when we reach true optimality.

Primal Pivoting



Now that we have our complete flow diagram containing all necessary information for the primal simplex method, let us understand the strategy for a pivot. In the above diagram, two dual infeasible arcs were identified, d → i and e → d. Let us choose d → i since it has a larger negative value.

Recalling that our basis is definitionally a spanning tree which means that any entering arc will form a cycle in our network. Pictured here, we see the dark blue arc which is our entering arc. Within the cycle there are two types of arcs, positive and negative. Positive arcs in the cycle point in the same direction as the entering arc, whereas negative arcs point in the opposite direction.

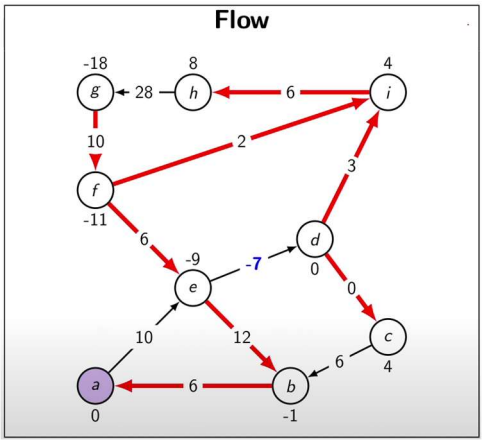


Let us consider this figure here. We note that the arc v6 → v1 enters and will enter with a primal flow value of t. In order to balance this cycle, the following results fall out. In order for v6 to have a supply of 0, it must send t units to v1. That node already had a flow of f1, so it must now send its

original f1 as well as the t it just received along to v2. Now, v2 is already receiving f1 and f2. We can think of this as v2 asking v3 “send me t units fewer than what you were sending me before so I can be balanced”. This pattern continues until you get this flow diagram of the cycle above.

Great, we have a non-feasible dual arc that will enter our basis and become primal feasible. We must now find an arc to exit. Recall that a non-primal feasible arc has a flow value of 0. This means we must choose the value of t such that one of the orange candidate arcs is 0. See that t = f2, then the arc v3 → v2 will be 0 and leave the basis, while preserving the overall balance of the network. So, this means after an arc enters the network to create a cycle, we choose the opposite direction arc with the smallest value.

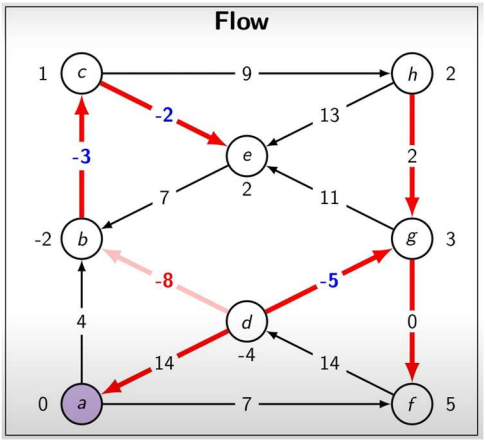
Take our example from above where we will allow d → i to enter the basis. Out of the candidate orange arcs to leave the basis. We see that either arc d → c or c → b can exit the basis. Note that this is a degenerate solution, and the choice is arbitrary. Since t = 3, we will recompute the primal flow variables. Arcs f → e and e → b will add 3 to their magnitudes. Arcs f → i, d → c, and c → b will subtract 3. Note that c → b will exit the network and become non-basic and d → c will still be in the basis but have a value of 0. Note that recomputing all of the primal flows, dual slack, and dual optimization is non-trivial but has been covered in detail before, so we leave this exercise to the reader. There are also some tricks to make the recomputation quicker by hand, but that falls outside of the scope of this paper.



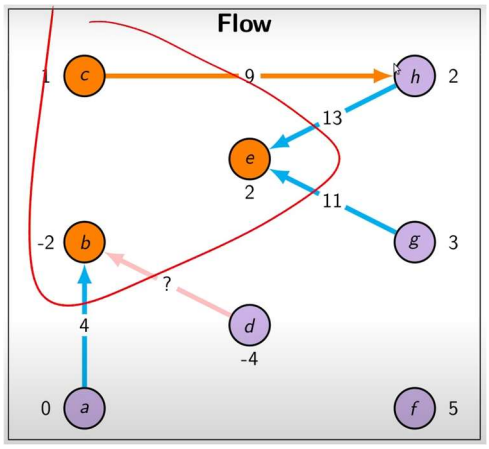
Dual Pivot

Here is the problem used to illustrate dual pivoting. A few things to note in this problem. Firstly, this flow network is dual optimal. We know this because all of the dual slack values on non-basic arcs are non-negative. We should also note that this network is definitely not primal feasible because of the several basic arcs that are negative.

The first step in the dual pivoting strategy is to identify the basic arc leaving. In this case we see that d → b should be removed since it is the most negative



value. After removing this arc, we see that our spanning tree is split and there is a separate subgraph. Then, a candidate for an entering arc is simply one that connects the sub graphs and maintains a spanning tree. In this case, see the arcs that cross the red triangular circle defining the disconnected subgraph. An arc with the same direction as the exiting arc will enter and will send the flow out. In this case, c → h will enter the basis and be sending 8 units out.



Before pivoting:

- The leaving arc (u, v) will be primal infeasible: $x_{uv} < 0$.
- The entering arc (s, t) will be dual feasible: $d_{st} \geq 0$.

After pivoting:

- The leaving arc (u, v) must have $x_{uv} = 0$ and $d_{uv} \geq 0$.
- The entering arc (s, t) must have $d_{st} = 0$ and ideally $x_{st} \geq 0$.
- The new basis must be dual feasible.

Two phase solver method

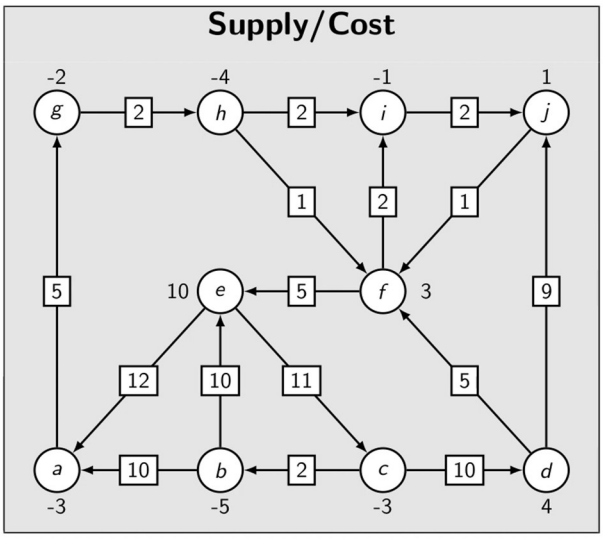
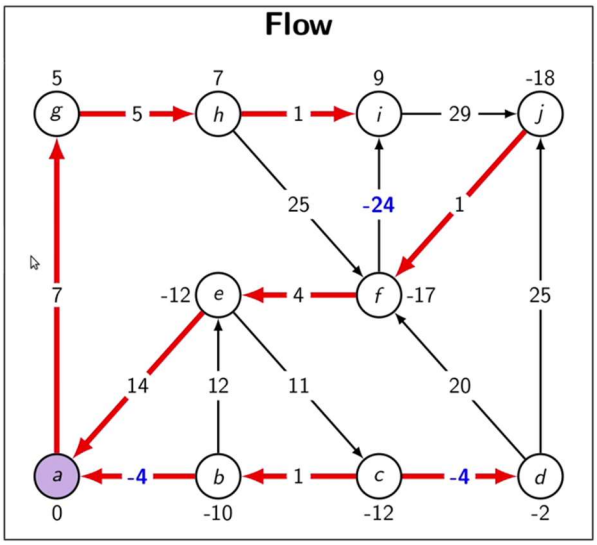
Now that the concepts of primal and dual pivoting have been established and outlined, a generalized network simplex method can be done. The general idea is to solve for a feasible primal or dual basis and then use that basis to solve the other problem. When you have a basis that solves both problems, you are guaranteed the optimal solution by strong duality.

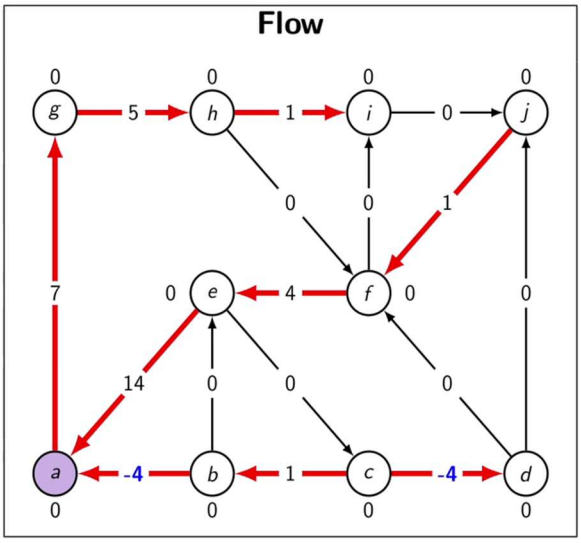
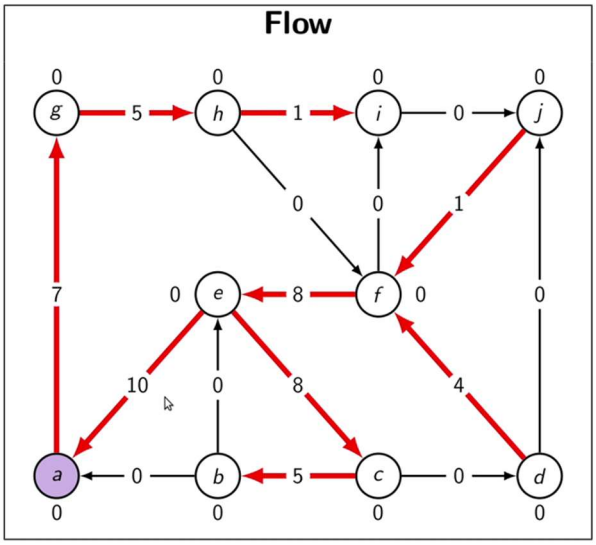
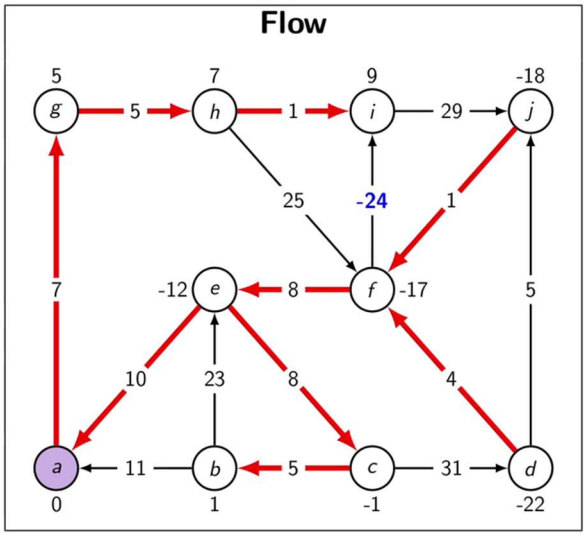
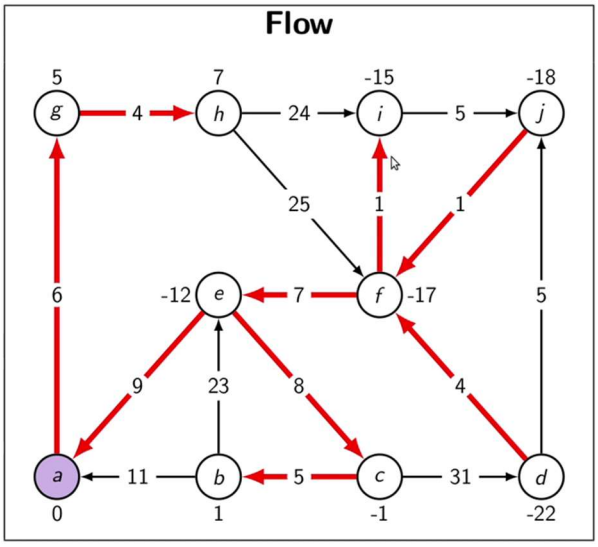
Let us lay out the dual first simplex method. Given a network G, create a network such that any spanning tree is dual feasible. We know that a dually feasible basis is when all non-basic arcs are non-negative. We construct G' to be the supply and cost network G, where cost arcs are set to 0. This way, any basis must be dual feasible since the dual slack values for the non-basic arcs will be 0. Let us call this transformed basis B'.

We take B', a dual feasible and (likely) primal infeasible basis, and drop it into G'. We solve the basis using primal pivots so that B' is both primal and dual feasible in G'. Then, transform B' back into G, noting that B' is now primal feasible (solved in the previous stage) but likely not dual feasible back in G. Next, taking the primally feasible B', solve with the dual simplex method on graph G such that

you result in a primal and dual feasible basis B which corresponds to the optimal solution.

Note that there is a primal first simplex method that follows the exact same steps, with some slight modifications. In this method, G' has all of its vertex supplies set equal to 0 so that all of the primal flow values are 0 and the basis vector B' will be primally feasible in G'.

Here is the initial supply and cost network to set up the problem. An algorithm is performed to generate a spanning tree solution, regardless of primal or dual feasibility.	Here is the initial basis. Looking at the blue numbers, we see it is neither prime nor dual feasible. Two of the primal flow values are negative on arcs b → a and c → d. This basis is also dual infeasible because of the negative dual slack value on the arc f → i.
	

After setting all cost values on every arc to 0, we see that our initial basis B' is pictures on the graph G' below. The dual pivoting simplex method will now be performed to make B' dual feasible in G'.	Here is the matching flow diagram after the dual pivoting technique has been finished. The basis B' is now dual feasible in G' and will be translated back to G.
	
After solving B' so that it was dual feasible in G', it is now placed into G. We see that the basis B' is not dual feasible because of the arc f → i and the primal pivoting strategy must be performed.	Here is our final basis B that is both primal and dual feasible in G. This is our optimal basis vector.
	

Case Study 9.1 p420

Introduction

Jake Nguyen, the manager of Asian foreign investment for Grant Hill Associates, faces a financial disaster due to a sudden collapse in the Japanese market, which has triggered a broader East Asian financial crisis. Despite prior warnings, Jake had significantly increased the firm's investment in Japan, raising the stake from $2.5 million to $15 million just one month before the crisis. At the time of investment, the exchange rate was 1 USD to 80 JPY, but after the devaluation, the rate has shifted to 1 USD to 125 JPY, leading to massive losses upon conversion back to U.S. dollars.

In response to the crisis, Grant Hill, the firm's president, orders Jake to immediately liquidate all holdings in Japan, Indonesia, and Malaysia and transfer the funds into U.S. bonds. However, the process is complicated by three factors. Firstly, the Japanese yen has sharply depreciated, significantly reducing the value of the firm's investments when converted back to U.S. dollars. Second, different banks charge varying fees for currency exchanges, increasing the importance of choosing the most cost-effective conversion path. Lastly, East Asian governments have placed strict limits on how much foreign currency can be withdrawn from their economies to prevent further financial instability.

To address this problem, Jake must develop an optimal strategy to liquidate and transfer the firm's holdings, minimizing losses due to exchange rates, transaction costs, and withdrawal limits. I will aim to solve his issue and give a minimized cost solution using techniques for network optimization.

Jake starts with holdings in Yen, Rupiah, and Ringgit. In order to correctly formulate the minimum cost flow problem such that the supply flow = demand flow, we convert the three supply currencies to USD equivalent. Jake holds $9.6mil Yen, $1.68mil Rupiah, and $5.6mil Ringgit. This means that the optimal solution should result in Jake receiving 16.68mil (the sum of the three supply amounts).

Solving with Excel

Formulating this problem in excel, our objective function will be minimizing the transaction cost for all of the transactions through the network. The amount sent across each arc multiplied by its unit cost (percentage of amount transferred) will be summed up and minimized.

For the starting network in part B, Jake should convert the equivalent of $2 million from Yen into four currencies - US dollars, Canadian dollars, Euros, and British pounds, distributing the amount equally among them. He also needs to exchange $1.6 million worth of Yen into Mexican pesos. For Rupiah conversions, he must transfer $200,000 worth into each of three currencies: US dollars, Canadian dollars, and Mexican pesos, while converting $1 million worth into Euros and $80,000 worth into British pounds. Additionally, from Ringgit, he should exchange $1.1 million worth into US dollars, $2.5 million into Euros, and $1 million each into British pounds and Mexican pesos. Finally, he must convert all previously obtained Canadian dollars, Euros, British pounds, and Mexican pesos back into US dollars, specifically amounts equivalent to $2.2 million in Canadian dollars, $5.5 million in Euros, $3.08 million in British pounds, and $2.8 million in Mexican pesos. After covering the $83,380 transaction costs from his American bank account, Jake will have $16796170 available to invest in the United States.

Source	Target	Sent		Capacity	Percent
Yen	Rupiah	0	<=	5000	0.50%
Yen	Ringgit	0	<=	5000	0.50%
Yen	US $	2000	<=	2000	0.40%
Yen	Can $	2000	<=	2000	0.40%
Yen	Euro	2000	<=	2000	0.40%
Yen	Pound	2000	<=	2000	0.25%
Yen	Peso	1600	<=	4000	0.50%
Rupiah	Yen	0	<=	5000	0.50%
Rupiah	Ringgit	0	<=	2000	0.70%
Rupiah	US $	200	<=	200	0.50%
Rupiah	Can $	200	<=	200	0.30%
Rupiah	Euro	1000	<=	1000	0.30%
Rupiah	Pound	280	<=	500	0.75%
Rupiah	Peso	0	<=	200	0.75%
Ringgit	Yen	0	<=	3000	0.50%
Ringgit	Rupiah	0	<=	4500	0.70%
Ringgit	US $	1100	<=	1500	0.70%
Ringgit	Can $	0	<=	1500	0.70%
Ringgit	Euro	2500	<=	2500	0.40%
Ringgit	Pound	1000	<=	1000	0.45%
Ringgit	Peso	1000	<=	1000	0.50%
Can $	US $	2200		-	0.05%
Can $	Euro	0		-	0.20%
Can $	Pound	0		-	0.10%
Can $	Peso	0		-	0.10%
Euro	US $	5500		-	0.10%
Euro	Can $	0		-	0.20%
Euro	Pound	0		-	0.05%
Euro	Peso	0		-	0.50%
Pound	US $	3280		-	0.10%
Pound	Can $	0		-	0.10%
Pound	Euro	0		-	0.05%
Pound	Peso	0		-	0.50%
Peso	US $	2600		-	0.10%
Peso	Can $	0		-	0.10%
Peso	Euro	0		-	0.50%
Peso	Pound	0		-	0.50%

Nodes	Net Flow		Supply/Demand	
Yen	9600	=	9600	=K15
Rupiah	1680	=	1680	=K16
Ringgit	5600	=	5600	=K17
Can $	0	=	0	
Euro	0	=	0	
Pound	0	=	0	
Peso	0	=	0	
US $	-16880	=	-16880	=SUM(K15:K17)

SUM:	83.38	=SUMPRODUCT(C2:C38, D2:D38)

	Starting Supply		USD Equivalent	
Yen	1200000	0.008	9600	=I15*J15
Rupiah	10500000	0.00016	1680	=I16*J16
Ringgit	28000	0.2	5600	=I17*J17

Nodes

=SUMIF(Source, Yen, Sent) - SUMIF(Target, Yen, Sent)

=SUMIF(Source, Rupiah, Sent) - SUMIF(Target, Rupiah, Sent)

=SUMIF(Source, Ringit, Sent) - SUMIF(Target, Ringit, Sent)

=SUMIF(Source, Can $, Sent) - SUMIF(Target, Can $, Sent)

=SUMIF(Source, Euro, Sent) - SUMIF(Target, Euro, Sent)

=SUMIF(Source, Pound, Sent) - SUMIF(Target, Pound, Sent)

=SUMIF(Source, Peso, Sent) - SUMIF(Target, Peso, Sent)

=SUMIF(Source, US $, Sent) - SUMIF(Target, US $, Sent)

Upon *removing transaction limits*, we see that the cost to convert all of the holdings decreases to $67,480. Jake should convert all 9.6million Yen to Pound to US, all 1.68million Rupiah to Canadian to US, and all 5.6million Ringgit to Euro to US. The elimination of capacity constraint yields this solution and transaction cost reduction for Jake.

Source	Target	Sent	Percent
Yen	Rupiah	0	0.50%
Yen	Ringgit	0	0.50%
Yen	US $	0	0.40%
Yen	Can $	0	0.40%
Yen	Euro	0	0.40%
Yen	Pound	9600	0.25%
Yen	Peso	0	0.50%
Rupiah	Yen	0	0.50%
Rupiah	Ringgit	0	0.70%
Rupiah	US $	0	0.50%
Rupiah	Can $	1680	0.30%
Rupiah	Euro	0	0.30%
Rupiah	Pound	0	0.75%
Rupiah	Peso	0	0.75%
Ringgit	Yen	0	0.50%
Ringgit	Rupiah	0	0.70%
Ringgit	US $	0	0.70%
Ringgit	Can $	0	0.70%
Ringgit	Euro	5600	0.40%
Ringgit	Pound	0	0.45%
Ringgit	Peso	0	0.50%
Can $	US $	1680	0.05%
Can $	Euro	0	0.20%
Can $	Pound	0	0.10%
Can $	Peso	0	0.10%
Euro	US $	5600	0.10%
Euro	Can $	0	0.20%
Euro	Pound	0	0.05%
Euro	Peso	0	0.50%
Pound	US $	9600	0.10%
Pound	Can $	0	0.10%
Pound	Euro	0	0.05%
Pound	Peso	0	0.50%
Peso	US $	0	0.10%
Peso	Can $	0	0.10%
Peso	Euro	0	0.50%
Peso	Pound	0	0.50%

Nodes	Net Flow		Supply/Demand	
Yen	9600	=	9600	=I15
Rupiah	1680	=	1680	=I16
Ringgit	5600	=	5600	=I17
Can $	0	=	0	
Euro	0	=	0	
Pound	0	=	0	
Peso	0	=	0	
US $	-16880	=	-16880	=SUM(I15:I17)

SUM:	67.48	=SUMPRODUCT(C2:C38, D2:D38)

	Starting Supply		USD Equivalent	
Yen	1200000	0.008	9600	=G15*H15
Rupiah	10500000	0.00016	1680	=G16*H16
Ringgit	28000	0.2	5600	=G17*H17

Nodes

=SUMIF(Source, Yen, Sent) - SUMIF(Target, Yen, Sent)

=SUMIF(Source, Rupiah, Sent) - SUMIF(Target, Rupiah, Sent)

=SUMIF(Source, Ringit, Sent) - SUMIF(Target, Ringit, Sent)

=SUMIF(Source, Can $, Sent) - SUMIF(Target, Can $, Sent)

=SUMIF(Source, Euro, Sent) - SUMIF(Target, Euro, Sent)

=SUMIF(Source, Pound, Sent) - SUMIF(Target, Pound, Sent)

=SUMIF(Source, Peso, Sent) - SUMIF(Target, Peso, Sent)

=SUMIF(Source, US $, Sent) - SUMIF(Target, US $, Sent)

When *increasing all transaction actions by 500% on Rupiah*, the amounts sent remain the same as the last part because there are no capacity constraints, but we notice that the transaction cost increases to $92,680. We see this 500% increase in the trading cost highlighted in the grey box.

Source	Target	Sent	Percent
Yen	Rupiah	0	0.50%
Yen	Ringgit	0	0.50%
Yen	US $	0	0.40%
Yen	Can $	0	0.40%
Yen	Euro	0	0.40%
Yen	Pound	9600	0.25%
Yen	Peso	0	0.50%
Rupiah	Yen	0	3.00%
Rupiah	Ringgit	0	4.20%
Rupiah	US $	0	3.00%
Rupiah	Can $	1680	1.80%
Rupiah	Euro	0	1.80%
Rupiah	Pound	0	4.50%
Rupiah	Peso	0	4.50%
Ringgit	Yen	0	0.50%
Ringgit	Rupiah	0	0.70%
Ringgit	US $	0	0.70%
Ringgit	Can $	0	0.70%
Ringgit	Euro	5600	0.40%
Ringgit	Pound	0	0.45%
Ringgit	Peso	0	0.50%
Can $	US $	1680	0.05%
Can $	Euro	0	0.20%
Can $	Pound	0	0.10%
Can $	Peso	0	0.10%
Euro	US $	5600	0.10%
Euro	Can $	0	0.20%
Euro	Pound	0	0.05%
Euro	Peso	0	0.50%
Pound	US $	9600	0.10%
Pound	Can $	0	0.10%
Pound	Euro	0	0.05%
Pound	Peso	0	0.50%
Peso	US $	0	0.10%
Peso	Can $	0	0.10%
Peso	Euro	0	0.50%
Peso	Pound	0	0.50%

Nodes	Net Flow		Supply/Demand	
Yen	9600	=	9600	=I15
Rupiah	1680	=	1680	=I16
Ringgit	5600	=	5600	=I17
Can $	0	=	0	
Euro	0	=	0	
Pound	0	=	0	
Peso	0	=	0	
US $	-16880	=	-16880	=SUM(I15:I17)

SUM:	92.68	=SUMPRODUCT(C2:C38, D2:D38)

	Starting	Rate	USD Equivalent	
Yen	1200000	0.008	9600	=G15*H15
Rupiah	10500000	0.00016	1680	=G16*H16
Ringgit	28000	0.2	5600	=G17*H17

Nodes

=SUMIF(Source, Yen, Sent) - SUMIF(Target, Yen, Sent)

=SUMIF(Source, Rupiah, Sent) - SUMIF(Target, Rupiah, Sent)

=SUMIF(Source, Ringit, Sent) - SUMIF(Target, Ringit, Sent)

=SUMIF(Source, Can $, Sent) - SUMIF(Target, Can $, Sent)

=SUMIF(Source, Euro, Sent) - SUMIF(Target, Euro, Sent)

=SUMIF(Source, Pound, Sent) - SUMIF(Target, Pound, Sent)

=SUMIF(Source, Peso, Sent) - SUMIF(Target, Peso, Sent)

=SUMIF(Source, US $, Sent) - SUMIF(Target, US $, Sent)

Citations

Bill Bird, Linear Programming - Lecture 15 - The Network Simplex Method: Initial Impressions. YouTube video. Accessed March 13, 2025. https://www.youtube.com/watch?v=HLXI-pk1Plw.

Bill Bird, Linear Programming - Lecture 16 - The Network Simplex Method: Graph Theoretic Interpretations. YouTube video. Accessed March 13, 2025. https://www.youtube.com/watch?v=ZdMM8xCptX4

Bill Bird, Linear Programming - Lecture 17 - The Network Simplex Method: Primal Pivoting. YouTube video. Accessed March 14, 2025. https://www.youtube.com/watch?v=zgtY5nGAMgY.

Bill Bird, Linear Programming - Lecture 18 - The Network Simplex Method: Dual Pivoting and Two Phase Methods. YouTube video. Accessed March 15, 2025. https://www.youtube.com/watch?v=ife2d0p4dug.

Hillier, Frederick S., and Gerald J. Lieberman. *Introduction to Operations Research*. 9th ed. New York: McGraw-Hill Higher Education, 2010.