CSCI 5654 (Fall 2011): Network Simplex Algorithm for Transshipment Problems.

Sriram Sankaranarayanan

November 15, 2011

1 Introduction

We will present the basic ideas behind the network Simplex algorithm for solving the transshipment problem. More details are available by reading chapter 19 from Chvatal's book or chapters 13 & 14, from Vanderbei's book. The exposition in this note will center around a simple example and introduce the various concepts. Going through the presentation in one of the books is very highly encouraged.

2 Transshipment Problem

Consider the operation of a company XYZ inc. that ships bananas from various parts of the country where they grow to parts where they are in demand. Of course, instead of banana's we could consider the shipment of oil across the country in a more practical scenario. The goods under question are supplied in a few places where they are available (either through import or local production) and need to be shipped to places where there is demand. The shipping of these goods is over a network.

Figure 1 shows an example of a network used to ship bananas across the country. There are 7 locations across the country. The demand/supply of bananas at each location is given:

Location	Miami	NY	Nashville	Wichita	Omaha	Denver	Boulder
Supply	100	200	0	0	0	0	0
Demand	0	0	0	10	40	20	230

The goal of the overall problem is to ship goods from Miami and NY where there is supply to places such as Wichita, Omaha, Denver and Boulder where the demand for bananas is especially high. We need to use the network in Figure 1 to ship bananas. Each edge in the network has an associated cost per unit of bananas shipped along the edge.

We note a few properties of the problem:

1. Each place has a demand for bananas or a supply. We assume that no place has both. If so, we can always cancel the local demand against the local supply and simply have the residual demand or supply for that node. This is quite trivial to ensure.

2. The sum total of demand equals the sum total of supply. In other words, all supply must be exhausted and all demand must be fulfilled. We will assume this for the basic problem but lift the relaxation later on.

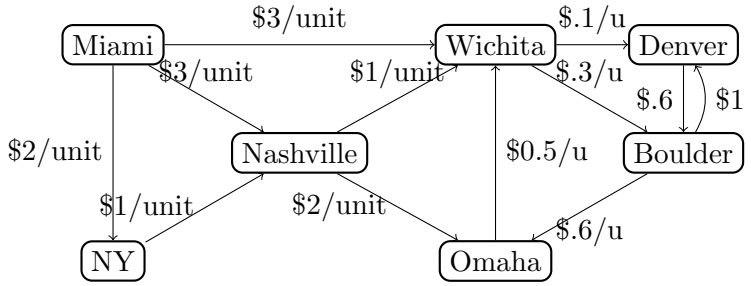


Figure 1: Example network for a transshipment problem with edge costs shown next to each edge.

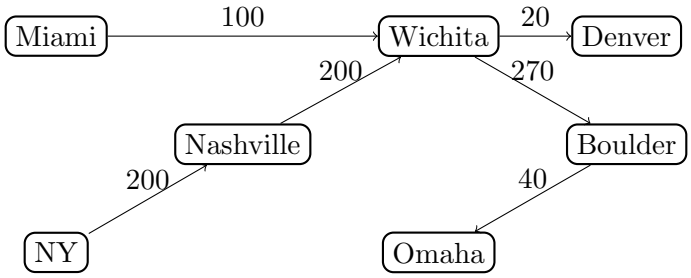


Figure 2: Optimal shipping schedule of bananas for network in Figure 1. The number on each edge shows how much banana is flowing. Flow along edges not shown is zero.

3. The flow along each edge in the network must respect the edge direction. We do not allow goods to flow in the reverse direction of an edge. In other words, for the example given, there can be no flow from Nashville to New York or from Wichita to Miami.

4. All edge costs are assumed to be non-negative.

Given a transshipment problem, we seek a shipping schedule that labels each edge with a non-negative flow such that the following constraints are satisfied:

1. Flow along each edge is non-negative (≥ 0).

2. Flow conservation holds for each node, i.e, incoming flow equals outgoing flow plus local demand (or minus local supply).

$$\text{Demand Node: } \sum_{e \in \mathsf{Incoming}(n)} \mathsf{Flow}(e) - \sum_{e \in \mathsf{Outgoing}(n)} \mathsf{Flow}(e) = \mathsf{Demand}(n)$$
$$\text{Supply Node: } \sum_{e \in \mathsf{Incoming}(n)} \mathsf{Flow}(e) - \sum_{e \in \mathsf{Outgoing}(n)} \mathsf{Flow}(e) = -\mathsf{Supply}(n)$$

Example 2.1. *Consider the shipping schedule in Figure 2. At Wichita, the total incoming flow is 300 and total outgoing is 290. The difference is the demand at Wichita which is* 10*.*

Likewise, at NY, the total incoming flow is 0 *and outgoing is* 200*. The difference is* -200 *which matches the supply of* 200 *at NY.*

The total cost of this shipping schedule is given by

$$200 \times 1 + 100 \times 3 + 200 \times 1 + 20.1 + 270 \times 0.3 + 40 \times 0.6 = 807$$

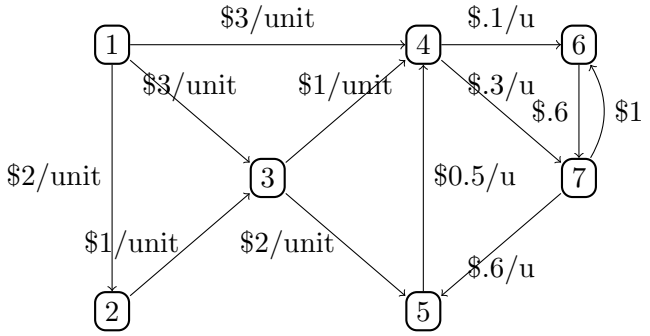


Figure 3: Example network for a transshipment problem with edge costs shown next to each edge.

	(1,2)	(1,3)	(1,4)	(2,3)	(3,4)	(3,5)	(4,6)	(4,7)	(5,4)	(6,7)	(7,6)	(7,5)
1 :	−1	−1	−1									
2 :	1			−1								
3 :		1		1	−1	−1						
4 :			1		1		−1	−1	1			
5 :						1			−1			1
6 :							1			−1	1	
7 :								1		1	−1	−1

Figure 4: Incidence matrix for network in Fig 3. For convenience, each row is labeled with its corresponding row and each column with the corresponding edge.

The output of the transshipment problem consists of two parts: (a) is there some shipping schedule that satisfies all demands and exhausts all supply while respecting the flow directions and (b) if so, what is the optimal schedule that minimizes cost.

If there is no possible shipping schedule the transshipment problem is infeasible. Transshipment problems by definition cannot be unbounded since there are no negative edge costs.

To summarize, the transshipment problem has the following specification.

Input: Network of nodes and edges, demand/supply at each node and shipping cost per unit for each edge.

Output: Optimal shipping schedule.

3 Transshipment Problem as a Linear Program

We will now show that a given transshipment problem can be modeled as a linear program.

Overall, the problem is defined by a graph G with nodes N and edges $E \subseteq N \times N$. We have a vector of demands at each node $\vec{b}$. We will denote a supply by a negative demand. Finally, we have a vector of costs for each edge $\vec{c}$.

Example 3.1. *Again, the problem in Figure 1 shows a graph with* 7 *nodes* $\{M, NY, N, W, O, D, B\}$ *with demand vector:*

$$\vec{b} = \begin{pmatrix} -100 & \leftarrow \textit{Miami} \\ -200 & \leftarrow \textit{NY} \\ 0 & \leftarrow \textit{Nashville} \\ 10 & \leftarrow \textit{Wichita} \\ 40 & \leftarrow \textit{Omaha} \\ 20 & \leftarrow \textit{Denver} \\ 230 & \leftarrow \textit{Boulder} \end{pmatrix}$$

For convenience, we will drop place names and refer to them by place numbers $1, 2, \ldots, 7$ *(Cf. Figure 3). The cost on each edge is shown in Figure 3. We write it down as a vector* $\vec{c}$ *after we have fixed an ordering of edges.*

We will now represent the graph G by means of an *incidence matrix* $A(G)$. The incidence matrix has a row for every node and a column for every edge. It represents each edge by placing a -1 for its source and $+1$ for its destination. The incidence matrix for the example network in Figure 3 is shown in Figure 4. Note that each column has two non-zero entries a -1 for the source node of the edge corresponding to the column and a $+1$ for the destination node. Looking along each row, we notice that a -1 entry in a row represents an outgoing edge and a $+1$ entry represents an incoming edge.

The decision variables for the transshipment problem are given by $\vec{x}$ where we have one entry x_e for each edge e. In our running example, the decision variables are

$$(x_{12}, x_{13}, x_{14}, x_{23}, x_{24}, x_{34}, x_{35}, x_{46}, x_{47}, x_{54}, x_{67}, x_{76}, x_{75}) .$$

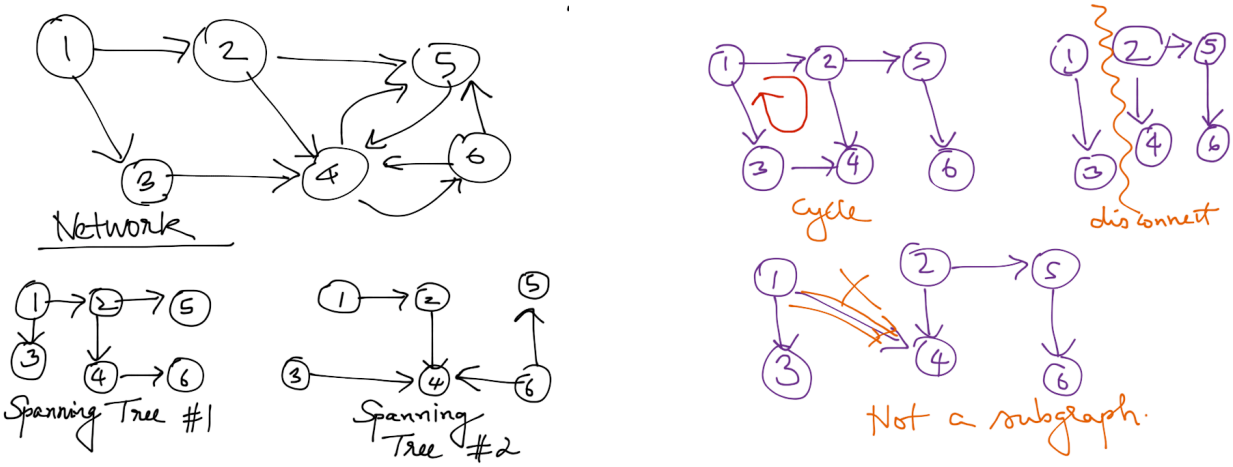
We normally write the vector $\vec{x}$ of decision variables as a column vector.

The overall transshipment problem can be written in matrix notation as:

$$\begin{array}{lllll} \min & \vec{c}^T\vec{x} & & & \leftarrow \text{ Total Cost} \\ \text{s.t.} & A\vec{x} & = & \vec{b} & \leftarrow \text{ Flow Conservation} \\ & \vec{x} & \geq & 0 & \leftarrow \text{ Directionality of flow.} \end{array}$$

Example 3.2. *As an example, we write out the constraints for the transshipment problem that has been our running example thus far (Fig. 3).*

$$\begin{array}{lll} \min & 2x_{12} + 3x_{13} + 3x_{14} + 2x_{23} + x_{34} + 2x_{35} + 0.3x_{47} + 0.1x_{46} + 0.5x_{54} + 0.6x_{67} + x_{76} & (*\textit{COST}*) \\ \text{s.t.} & -x_{12} - x_{13} - x_{14} = -100 & (*\textit{Conservation for node 1}*) \\ & x_{12} - x_{23} = -200 & (*\textit{Conservation for node 2}*) \\ & x_{13} + x_{23} - x_{34} - x_{35} = 0 & (*\textit{Conservation for node 3}*) \\ & x_{14} + x_{34} + x_{54} - x_{46} - x_{47} = 10 & \\ & x_{35} + x_{75} - x_{54} = 40 & \\ & x_{46} + x_{76} - x_{67} = 20 & \\ & x_{47} + x_{67} - x_{76} - x_{75} = 230 & (*\textit{Conservation for node 7}*) \\ & x_{12}, x_{13}, \ldots, x_{76} \geq 0 & (*\textit{Directionality for flow}*) \end{array}$$



(a) (b)

Figure 5: An example network (left) with some spanning trees and (right) non-examples of spanning trees.

4 Network Simplex Algorithm

We will now describe the network simplex algorithm. Instead of dictionaries, this version operates on *spanning trees*.

Definition 4.1. *A tree is an undirected graph that is (a) fully connected and (b) has no cycles.*

Given a directed graph G, a spanning tree is a subset of edges of the graph such that if we treat the edges as undirected, the resulting subgraph is a tree. Figure 5 shows some examples of spanning trees for a given network and some non-examples as well. Note that for a graph with n nodes, any spanning tree involves exactly $n-1$ edges. You may verify this in Figure 5. The original graph has 6 nodes and each spanning tree has 5 edges.

Network simplex method maintains a spanning tree at each iteration. It works as follows:

1. Land on an initial feasible spanning tree. If this fails, we declare the problem as infeasible.
2. Pivot from one spanning tree to next.
3. Until spanning tree is optimal.

The optimal spanning tree yields the optimal shipping schedule. As we did for regular Simplex, we will first describe the pivoting process and then talk about initialization later.

4.1 Feasible Spanning Trees

First, given a spanning tree T of a network, we assume that only edges on the tree carry flow. Edges off the tree do not carry any flow. With this assumption, we find that given a spanning tree, it is possible to uniquely work out the value of flow on the edges of the tree that will satisfy flow constraints at each node.

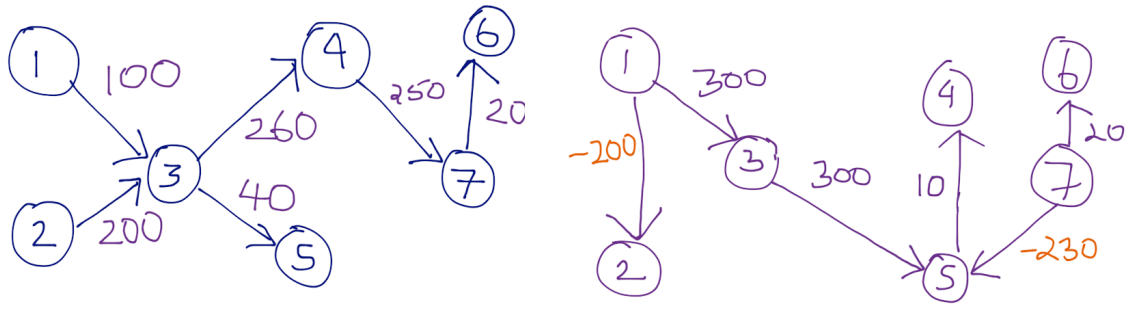


Figure 6: (left) A feasible spanning tree for network in Fig. 3 and (right) an infeasible spanning tree for the same network.

Example 4.1. *Figure 6 shows two example spanning trees T_1 and T_2 for the network in Figure 3. Let us work out the flows on the edges for each tree.*

We begin with tree T_1. We note that nodes $1, 2, 4, 6$ are leaves of the tree. A leaf simply has one edge incident on it. Therefore, we note that the flows on each of the nodes must simply match be the supply on the node or the demand.

$$x_{13} = 100,\ x_{23} = 200,\ x_{67} = 20,\ x_{45} = 40\,.$$

This gives us enough data to work out x_{34} to satisfy the flow constraint on 3:

$$x_{13} + x_{23} - x_{34} - x_{35} = b_3 = 0$$

This immediately gives us $x_{45} = 260$. This now gives us enough information to find x_{47}.

$$x_{47} = 250$$

We note that all flows worked out to be non-negative and all flow conservation constraints are satisfied.

Let us do the same for tree T_2. The leaves are $2, 4, 6$.

At node 2, we have one edge entering and a supply in 2 of 200. Therefore, the only way to satisfy flow constraint is

$$x_{12} = b_2 = -200\,.$$

Likewise, we obtain

$$x_{54} = b_5 = 10,\quad x_{76} = b_6 = 20$$

based on the demands at the leaf nodes 4 and 6. Knowing x_{12} allows us to work out x_{13}:

$$-x_{12} - x_{13} = b_1 = -100\,.$$

Therefore, we get

$$x_{13} = 300$$

Likewise, we can solve the x_{35} and x_{57}, separately, by invoking flow conservation constraints at nodes 3 and 7 respectively to get,

$$x_{35} = 300, x_{57} = -230\,.$$

Note however, that the unique solution we get from flow conservation for T_2 violates the directionality of the flow. Therefore, T_2 is called an infeasible tree.

A spanning tree T is feasible if the unique values of edge flows on the spanning tree that satisfy all flow conservation laws also respects all the flow constraints.

Network simplex is initialized to some feasible spanning tree.

5 Node Shadow Prices

The next concept is that of a shadow price per unit at each node. Let us assume that T_1 (shown to the left) in Figure 6 is the current feasible schedule for company XYZ.

We recall the cost on each spanning tree edge:

Edge	$(1,3)$	$(2,3)$	$(3,4)$	$(3,5)$	$(4,7)$	$(7,6)$
Cost (\$/unit)	3	1	1	2	0.3	1

Q: What price must XYZ charge at each location given that the shipping schedule is along the spanning tree T_1 (left) in Figure 6, so that it breaks even?

Note that the breaking even part is key. Alternatively, we could have said what is the minimum price it can charge and not suffer a loss (same thing). The answer is highly dependent on choosing a reference price at a node. Just for reference, we will say:

XYZ charges \$ 1 at node 1 for 1 unit of bananas.

If XYZ charges \$ 1 at node 1,

1. It must charge, \$ 1 + 3 = \$ 4 units at node 3. Why? Well, it charges \$ 1 at node 1. It costs \$ 3 to ship to 3 across the link (1,3). So it must charge \$ 4 at node 3.
2. It must charge \$ 3 units at node 2. Why \$ 3? In this case, it can ship for \$1 across (2,3) edge and match the price at node 3 already established at \$ 4.
3. \$ 5 in node 4 (\$ 4 at node 3 and \$1 to ship across (3,4)).
4. \$ 6 in node 5.
5. \$ 5.3 in node 7 and
6. \$ 6.3 in node 6.

Note that the reference is chosen to be arbitrary node, arbitrary price. But once we choose it, we can immediately pin down the shadow prices at each node. Let $y_1, \ldots, y_n$ denote the shadow prices.

If this is the price that company XYZ has to charge at each node, the question is whether a competitor can cause loss to XYZ by buying at some node and shipping across an edge not on the spanning tree?

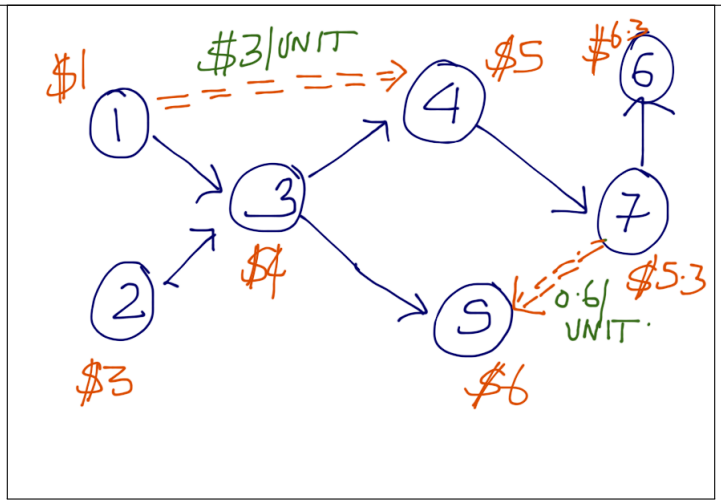


Figure 7: Spanning tree with shadow prices shown next to each node. Also two off spanning tree edges $(1,4)$ or $(7,5)$ can be used by a "competitor" to perform arbitrage if this schedule were to be used.

Figure 7 shows the shadow prices for each node. Also note that a competitor to XYZ could make illicit profits at the expense of XYZ by buying from XYZ for \$1 at node 1 and selling to the public at node 4 and shipping across edge $(1,4)$:

Shadow price at node $1(y_1)$ + cost of shipping across $(1,4)(c_{1,4})$ < shadow price at node $4(y_4)$

The same is true for $(7,5)$ edge. Note that the way shadow prices are constructed, the only way to make a profit for a competitor is to ship on an edge that is not part of the spanning tree.

Entering Edge: Let us fix a spanning tree T. Let $y_1, \ldots, y_n$ be the shadow prices at nodes $1, \ldots, n$. We call an edge (i,j) that is NOT in a spanning tree an entering edge, if

$$y_i + c_{i,j} < y_j$$

In other words, a competitor can buy at node i and ship along (i,j) to undersell the price at node j.

For the spanning tree in Figures 6 and Fig. 7, the possible entering edges are:

$$(3,4) : y_4 + c_{4,5} = 1 + 3 < 5 = y_5 \text{ and } (7,5) : \; y_7 + c_{7,5} = 5.3 + .6 < 6 = y_5 \, .$$

Are there any other entering edges?? Let us consider possible choices:

Edge	$(1,2)$	$(1,4)$	$(5,4)$	$(4,6)$	$(6,7)$	$(7,5)$
Entering?	No	Yes	No	Yes	No	Yes

It turns out that $(4,6)$ is one other choice of an entering edge.

Leaving Edge: Given the entering edge, we will now pivot as follows to find a *leaving* edge:

- Add the entering edge to the spanning tree and let us send a flow of $t \geq 0$ along this edge.

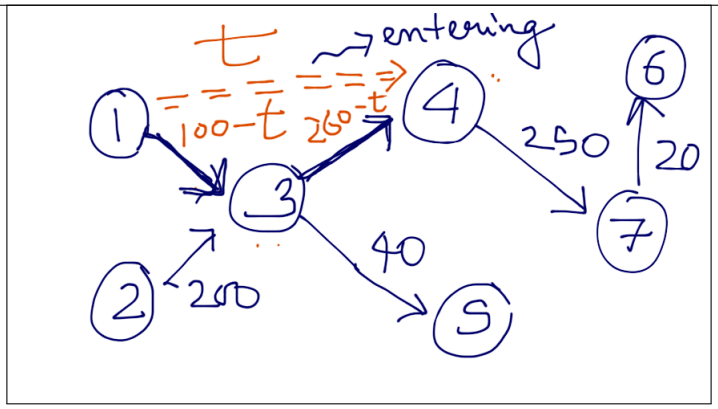


Figure 8: Leaving edge analysis for entering edge $(1, 4)$.

- Adjust flow along other edges to preserve flow constraints. Thus, we will also find the edge that constraints the value of t the most. This will become the leaving edge.
- The leaving edge leaves the spanning tree and entering edge will be added to the spanning tree. The result will be a feasible spanning tree.

Figure 8 shows the leaving edge analysis if the edge $(1, 4)$ is chosen as the entering edge.

1. Let us add $(1, 4)$ as the entering edge.
2. Its addition creates a directed cycle involving $(1, 4)$ and the edges (1,3) and (3,4). The leaving edge will be one of the latter two.
3. Let us add $t \geq 0$ units of flow to the edge $(1, 4)$. We are required to accommodate this extra flow by *modifying the flow on edges* $(1, 3)$ *and* $(1, 4)$.
4. The flow on other edges in the tree has to be unaltered.
5. To maintain flow conservation at 1, we conclude that edge $(1, 3)$ has to carry t units less and therefore $100 - t$ units of flow and edge $(3, 4)$ carries t units less as well to yield $260 - t$ units of flow.

From the analysis above, we find that as t increases, the flow along edges $(1, 3)$ and $(3, 4)$ decreases. The edge $(1, 3)$ will be the first edge to hit 0. It limits the increase of t to 100, whereas edge $(3, 4)$ limits increase to 260. Therefore $(1, 3)$ is the leaving edge.

Figure 9 shows the new spanning tree obtained. We first work out the flows on the edges so that the flow conservation laws are satisfied. No wonder this turns out to be feasible. Next, we arbitrarily set the price to be \$1 at node 5 and work out all the shadow prices at all the nodes.

Based on this there are two possible entering edges $(7, 5)$ or $(4, 6)$. Convince yourself that there are no other possible choices.

Next, we choose $(7, 5)$ as the entering edge. Figure 9 shows the leaving variable analysis. The entering edge creates a directed cycle that involves edges $(3, 4), (3, 5)$ and $(4, 7)$. One of these must leave to keep the spanning tree. Suppose an extra flow of $+t$ were added to the entering edge $(7, 5)$,

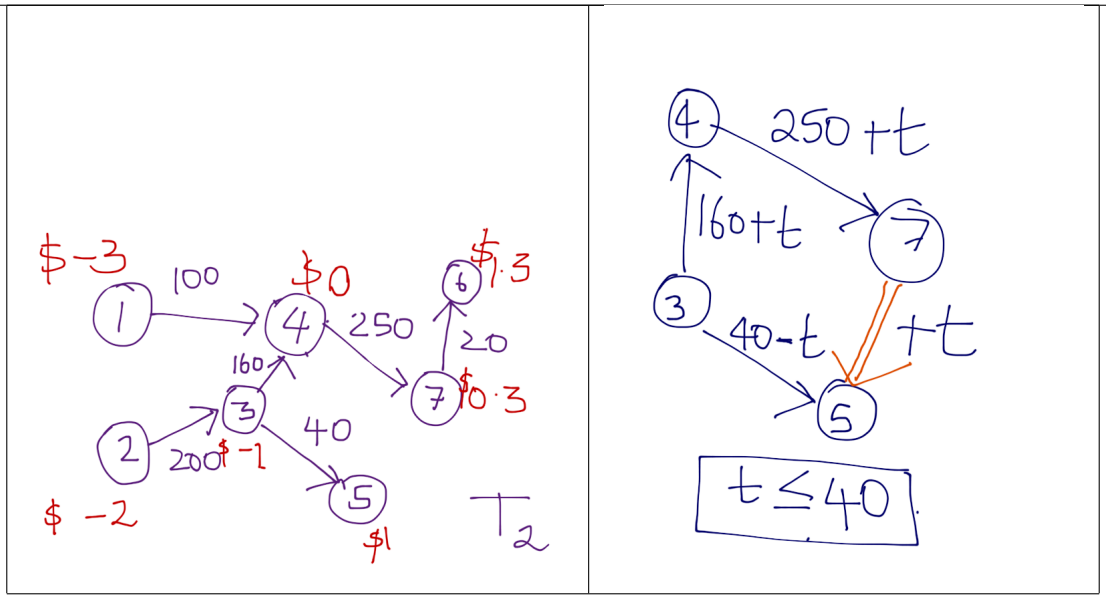


Figure 9: (a) The spanning tree T_2 after $(1,3)$ leaves with flows on edges and node shadow prices shown, and $(3,4)$ enters, and (b) leaving variable analysis when entering edge is $(7,5)$.

we note that extra flow of t leaves 7. So the flow on $(4,7)$ becomes $250+t$, the flow on $(3,4)$ becomes $160+t$. To compensate, the flow on $(3,5)$ becomes $40-t$.

This shows that $(3,5)$ is the leaving edge and $t \leq 40$. We have worked out the flows on edges and the node shadow prices using a reference of \$ 0 on node 1. The dotted edge $(4,6)$ is the only entering edge for this spanning tree.

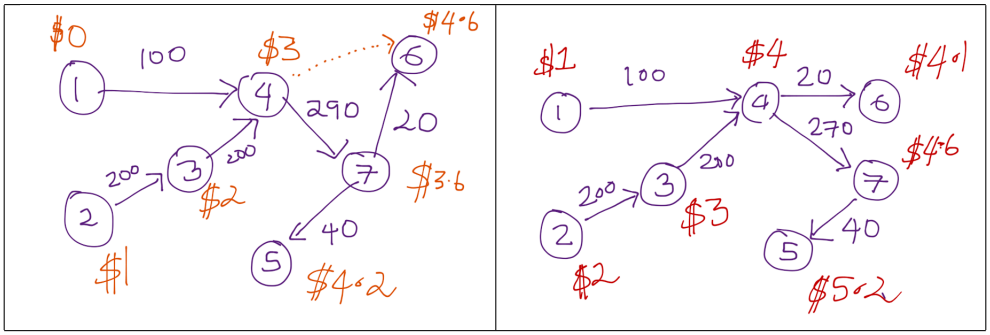


Figure 10: Spanning trees T_3 and T_4.

The next spanning tree T_3 is shown in Figure 10. We have worked out the node shadow prices and edge flows for T_3. We note that the edge $(4,6)$ is the only entering edge. Corresponding to this choice, you will find that $(7,6)$ is the leaving edge. This yields the spanning tree T_4 that is shown right in Figure 10. We notice that there is no choice for entering edge in T_4. Consequently this is the final spanning tree and thus the optimal solution.

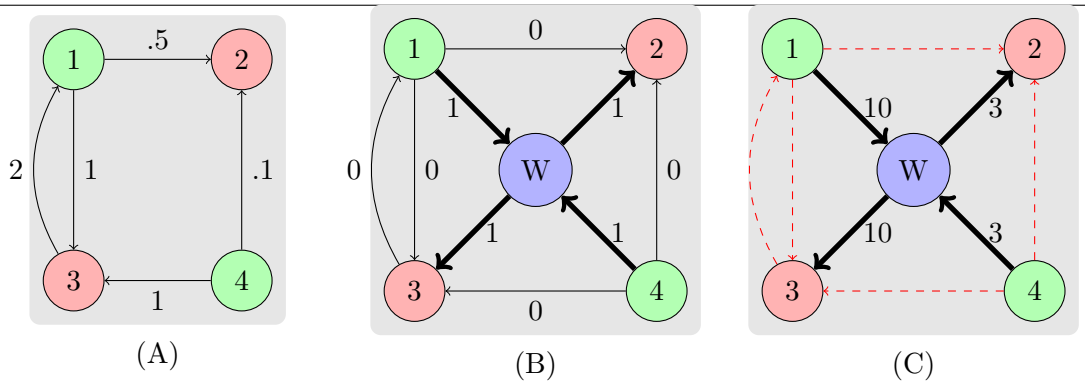


Figure 11: (A) Network for which initial feasible spanning tree is to be discovered with edge costs shown, (B) Adding a new artificial node and artificial edges(thick) with new edge costs and (C) the initial feasible spanning tree showing flows on edges.

6 Initialization

We will now discuss the process of initializing the network simplex algorithm to find a feasible spanning tree to start off with. Just as in the original simplex algorithm, our approach would be to solve an auxilliary LP with extra artificial edges and an artificial node. Further, the objective of the auxilliary LP is modified.

Adding Aritificial Node and Edges. To initialize, we setup an auxilliary problem by adding a new intermediate node w to the problem and adding the following artificial edges:

1. An artificial edge from each supply node ($b_i < 0$) to w.
2. An artificial edge from w to each demand node $b_i \geq 0$.

Note that the node w itself has no demand or no supply.

We update the costs for each edge so that original problem edges now have a cost of 0 and all artificial edges have a cost of 1.

As an example, Figure 11 shows a network with the following demand vector:

$$\begin{pmatrix} -10 \\ 3 \\ 10 \\ -3 \end{pmatrix}$$

and edge costs as shown in Fig. 11(A). Next, Fig. 11(B) shows the network for the initialization phase with artificial edges added. Note that each original problem edge cost is now 0 and artificial edges now cost 1.

Note that the addition of an artificial node W is not strictly necessary. It is possible to choose a original problem node x to serve the same purpose as W.

Initial Spanning Tree. With the artificial edges added, it is easy to get a feasible initial spanning tree by simply choosing all the artificial edges to be part of the spanning tree. If the demands and supplies in the original problem are equal then this spanning tree will always be feasible.

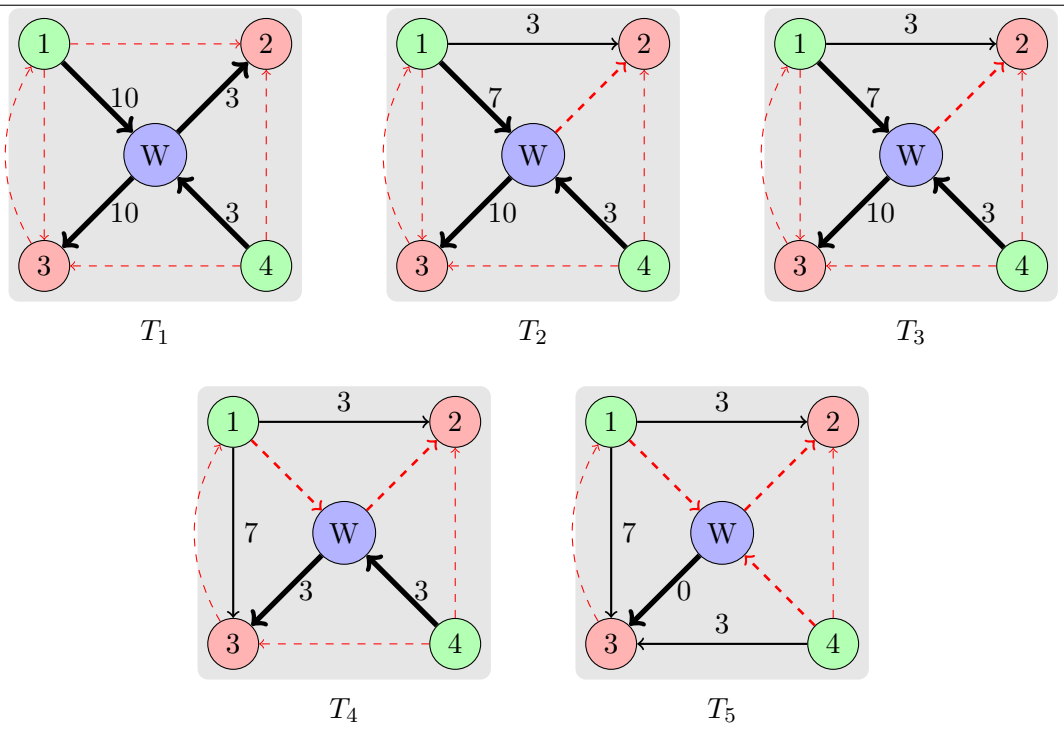


Figure 12: Sequence of Network Simplex steps for initialization phase on network in Fig. 11.

Figure 11(C) shows the spanning tree on the example.

We now solve network simplex to find the optimal solution for the auxilliary problem. The optimal cost for this problem can be zero or strictly positive.

Feasible: The original problem is feasible if and only if the optimal value found by the auxilliary problem is zero.

Infeasible: The original problem is infeasible if and only if the optimal value found by the auxilliary problem is strictly positive.

Now let us consider the example.

Let us consider the example from Figure 12. We obtain the steps shown in Figure 12. We note that the optimal cost is 0, indicating that the original problem is feasible. Furthermore, we notice that there is precisely one artificial edge in the spanning tree which carries no flow (i.e, the artificial node w is a leaf node). In this case, w may be removed to obtain an initial spanning tree obtained by edges $(1, 2)$, $(1, 3)$ and $(4, 3)$ carrying the flows shown in tree T_5 of Fig. 12. Of course, this is not optimal. We completely remove all artificial edges and node w to obtain an initial spanning tree from which Simplex may proceed.

This gives rise to a key question:

Is the artificial node guaranteed to be a leaf in the final spanning tree for aux. problem?

As we shall see presently, the answer is in general, no.

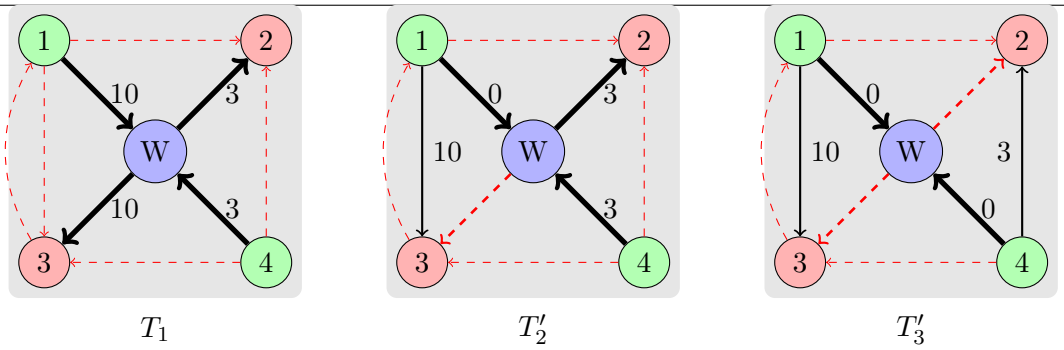


Figure 13: Alternative sequence of Network Simplex steps for initialization phase on network in Fig. 11.

Figure 13 shows an alternative series of steps that yield a spanning tree in which w is no longer a leaf. There are two artificial edges $(1, w)$ and $(4, w)$ in the tree and removing these with node w will not yield a spanning tree for the original problem. Instead it yields a *forest* with two edges $(1, 3)$ and $(4, 2)$.

The question is how do we deal with this situation?

Conceptually, we can proceed in one of two ways: (A) update the edge costs of the remaining artificial edges and simply keep them around in the spanning tree until they leave (at which point, we can delete them) or (B) systematically drive out all but one artificial edges from the spanning tree at which point, we can safely delete w.

Approach (B) only works if the original problem network was connected. If not, we will need the artificial node w to keep the spanning tree property or consider decomposing the original problem into subproblems designated by the connected components.

Let us consider approach (A) of re-costing the edges.

1. Restore the original problem costs to all problem edges.

2. Keep the artificial node w and any artificial edges that remain in the final spanning tree but increase their edge costs to ∞ for edges coming into w and $-\infty$ for edges leaving w.

During the new Simplex procedure, the artificial edges will simply remain in the tree and once they leave, they will never be a candidate for entering again. This is because, the price on an artificial node w will always be $+\infty$ (we assume that $x + \infty = \infty$ for all finite x).

Finally, whenever w becomes a leaf in one of the trees encountered, we will simply allow delete it.

Approach (A) still requires special handling of the node w. For instance, for the computation of the shadow prices, it is important that the reference cost be chosen as finite for a original problem node to ensure that shadow price for w is always ∞.

Let us consider approach (B) assuming that the original problem is connected. We will now consider the problem of driving out all artificial edges. If w is a leaf node, it can be deleted at once.

Take an edge (v_i, v_j) from the original problem such that

1. (v_i, v_j) is not part of the final spanning tree from the auxilliary problem.

2. The unique path (in the undirected sense) from v_i to v_j in the spanning tree passes through w.

As long as w is not a leaf and the original graph is connected (in the undirected sense), such an edge (v_i, v_j) can always be found.

We force (v_i, v_j) to be the entering edge and any one of the edges involving w on the path from v_i to v_j to leave the tree. The new edge (v_i, v_j) carries 0 flow. We can now check that flow balance will not be disturbed for any of the nodes in this process. Next, the spanning tree property is still maintained.