Applications of Network Flows

Jeffrey A. Appleget, Steven B. Horton

Introduction

A great variety of military problems can be modeled with network flows. This chapter will discuss two of the most basic network flow problems: *maximum flow* and *shortest path*. Before we can get to these military applications, however, it is important to understand some of the fundamental concepts of *linear programming* and how they relate to *integer programming*. If you are generally familiar with these concepts, you can skip the next section.

Linear and Integer Programming

A *linear program* is an optimization problem of the form

$$\begin{aligned} &\text{maximize } \sum_{j=1}^{n} c_j x_j \\ &\text{subject to: } \sum_{j=1}^{n} a_{ij} x_j \le b_i \quad (i = 1,2,\ldots,m) \\ &\qquad x_j \ge 0 \qquad\qquad (j = 1,2,\ldots,n). \end{aligned}$$

The x_i terms are the *decision variables* and the a_{ij}, b_i, and c_j terms are data that are typically part of the problem. $\sum_{j=1}^{n} c_j x_j$ is called the *objective function* and both $\sum_{j=1}^{n} a_{ij} x_j \le b_i$ and $x_j \ge 0$ are called *constraints*. Constraints of the form $x_j \ge 0$ are known as *nonnegativity constraints*. Note that in the standard form above there are *n* nonnegativity constraints and *m* of the other type.

Example

Giapetto's Woodcarving, Inc. manufactures and sells toy soldiers and toy trains. A soldier sells for $27 and a train sells for $21. We assume all soldiers and trains manufactured can be sold. Soldiers require 12 hours of labor and 2 units of wood. Trains require 3 hours of labor and 7 units of wood. For this week, Giapetto has 81 hours of labor and 111 units of wood available. How many soldiers and trains should he make to maximize revenue?

The first step in solving this type (and most other types!) of problem is to define your variables. If we let x_1 be the number of soldiers to make and x_2 be the number of trains to make, we can think of Giapetto's problem as the following linear program:

$$\begin{aligned} \text{maximize} \quad & 27x_1 + 21x_2 \\ \text{subject to:} \quad & 12x_1 + 3x_2 \le 81 \\ & 2x_1 + 7x_2 \le 111 \\ & x_1 \ge 0 \\ & x_2 \ge 0 \end{aligned}$$

The first line is the objective function. In this case it represents the money that Giapetto gets for each possible decision he might make. The first two constraints represent restrictions imposed by the limited supply of labor and wood, respectively, at Giapetto's disposal. The other two constraints are simply logical restrictions against building a negative number of soldiers or trains.

Although it will not be covered here, there are a number of ways to find the solution to linear programs. See [2] or [3] if you'd like to learn more about these methods. The solution to the example above is $x_1 = 3$ and $x_2 = 15$. Can you think of a way to find this solution? Look at the picture in figure 1 below of the region where each of the four inequalities is satisfied. This region of allowable solutions is called the *feasible region*. Does this help you see a way to solve this type of problem? How does the objective function get considered in your solution method?

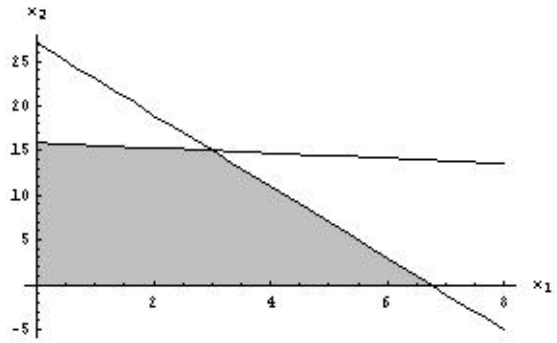

Figure 1: Feasible Region

Notice that the optimal solution of $x_1 = 3$ and $x_2 = 15$ "happens" to occur where two of the constraints intersect. Do you think this is a coincidence?

Something you might have noticed about the example above is the fact that we were lucky enough to have an *integral* optimal solution. An integral solution is one where each decision variable is an integer (whole number). This would have made it easy to tell Giapetto what to do: make 3 soldiers and 15 trains. Suppose we make a very small change in the problem and give Giapetto an extra hour of labor for a total of 82. Now the linear program is

$$\begin{aligned} \text{maximize} \quad & 27x_1 + 21x_2 \\ \text{subject to:} \quad & 12x_1 + 3x_2 \le 82 \\ & 2x_1 + 7x_2 \le 111 \\ & x_1 \ge 0 \\ & x_2 \ge 0 \end{aligned}$$

This small change has the effect of moving one of the constraints "out" slightly. You can again find the linear program solution graphically or with some other method, but it is unfortunately no longer integral: $x_1 = \frac{241}{78} \cong 3.0897$ and $x_2 = \frac{584}{39} \cong 14.9744$. While we can't tell Giapetto to make 3.0897 soldiers and 14.9744 trains, we can at least stick with our old solution of 3 soldiers and 15 trains and "waste" the extra hour of labor. There is no longer any assurance that (3,15) is the *best* solution, but it is at least a *feasible* solution, since it obviously still satisfies all of the constraints.

On the other hand, consider what happens if we lose an hour of labor as opposed to gaining one. Now our linear program is

$$\begin{aligned} \text{maximize} \quad & 27x_1 + 21x_2 \\ \text{subject to:} \quad & 12x_1 + 3x_2 \le 80 \\ & 2x_1 + 7x_2 \le 111 \\ & x_1 \ge 0 \\ & x_2 \ge 0 \end{aligned}$$

and the optimal linear program solution is $x_1 = \frac{227}{78} \cong 2.9103$ and $x_2 = \frac{586}{39} \cong 15.0256$. Now we have bigger problems. Not only is our new solution not integral, but our old friend and previous solution (3,15) is now not feasible since it violates the first constraint. So now even *finding* a feasible

solution seems to be a challenge. In fact, although linear programs are efficiently solvable, no efficient procedure is known to solve all *integer programming* problems. An integer program is simply a linear program in which the decision variables can only be integers. A *binary integer program* is a linear program where the decision variables can only take on the values 0 or 1.

There are several points here. Linear programs are easy to solve, but when our problem requires an integral solution in the real world, the linear programming model generally fails to give us what we need. However, when the linear programming solution happens to be integral, we know we have the right answer: we can go tell Giapetto what to do directly from this solution without any interpretation or other difficulties. Fortunately, there are several general classes of problems where under certain conditions the linear programming solution always works out to be integral. Among these are the two types of problems we will study next: maximum flows and shortest paths.

Modeling Military Maximum Flow Problems

You are the movement officer for an infantry division. The division must move from the port of debarkation to an assembly area in the corps rear area. Figure 2 is a model of the road network in your area. Your task is to get as many companies as possible to the assembly area in the first hour.

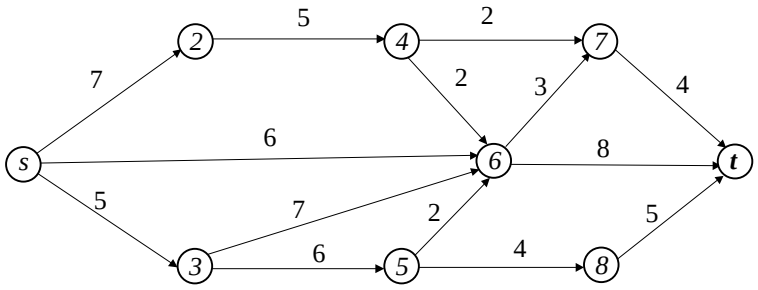

Figure 2: Maximum Flow Example

The starting node *s* represents the port of debarkation and the termination node *t* represents the assembly area. Other nodes represent road junctions. Arcs are identified by the nodes which they connect. Each arc represents a road, and each road has a capacity, c_{ij}, in companies per hour, as shown. For example, the arc (3,6) has a capacity of 7 companies per hour, while for arc (6,*t*), $c_{ij} = 8$.

We assume that the arcs are one-way, or directed arcs. We call the overall network $G=(N,A)$ where N is the set of all nodes $\{s,2,3,\ldots,n\text{-}1,t\}$ and A is the set of all existing arcs $\{(s,2),(s,3),(s,6),(3,6),\ldots,(6,t),(7,t),(8,t)\}$.

We can model this maximum flow problem using linear programming. Let x_{ij} represent the number of companies that travel on road (*i*,*j*). To simplify the model, we add the arc (*t*,*s*) and set $c_{ts} = \infty$.

$$\begin{aligned} &\max \ x_{ts} \\ &st \ \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0 \text{ for } i \in N \\ &\quad x_{ij} \le c_{ij} \text{ for all } (i,j) \in A \\ &\quad x_{ij} \ge 0 \text{ for all } (i,j) \in A. \end{aligned}$$

As we discussed in section II, this is a type of problem where solving the linear program will always give as an integral solution as long as each c_{ij} is an integer.

Modeling Military Shortest Path Problems

The shortest path problem is another simple network flow problem. As the name implies, the shortest path problem finds the shortest path between two points. For simplicity, we will consider shortest path problems that find the shortest path from some starting node *s* to a termination node *t*. Military applications include finding the shortest path for deploying a unit from some rear assembly area to a tactical assembly area and finding the most reliable route between two nodes.

Example: Deploying a unit

As the S-3 of the 1st Forward Support Battalion, you are tasked to find the shortest route from Assembly Area Alpha to Tactical Assembly Area Support. You are given a sketch of the road network in figure 3 below.

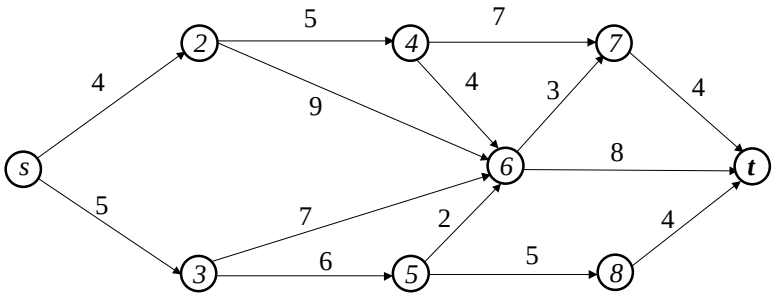

Figure 3: Shortest Path Example

First, all the nodes in the network represent road junctions where multiple roads intersect. We can assign numbers 2 through n-1 to number the nodes between s and t, giving us a total of n nodes. If a suitable road exists between two nodes, then we connect the nodes with an arc. Arcs are again identified by the nodes which they connect. Each arc will have a length associated with it, which we will call l_{ij}. We can formulate this problem as a linear programming problem where x_{ij} represents arc (i,j):

$$\begin{aligned}
&\min \sum_{(i,j)\in A} l_{ij} x_{ij} \\
&st \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 1 \text{ for } i = s \\
&\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = -1 \text{ for } i = t \\
&\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0 \text{ for all } i \in N - \{s\} - \{t\} \\
&x_{ij} \geq 0 \text{ for all } (i,j) \in A.
\end{aligned}$$

As was true for maximum flow problems, this is a type of problem where solving the linear program will always give as an integral solution. In this case, we can be assured that the values for x_{ij} will be either 0 or 1: 0 if the arc (i,j) is not in the shortest path, and 1 if arc (i,j) is in the shortest path.

Example: Finding the most reliable route

Now that the FSB has reached TAA Support, combat has begun. As the brigade that your battalion supports moves forward, you must move forward as well to effect timely support. Again, the road network leaves you with choices. This time you are concerned with finding the most reliable route. You will move in 6 hours. Because combat creates rapidly changing situations, you will not know which routes are open or closed, but rather you will be provided with the probability that each section of the road network (represented by an arc) is operational when you are to traverse it. Let p_{ij} be the probability that arc (i,j) is operational. We are now interested in solving the problem:

$$\begin{aligned}
&\max \prod_{(i,j):x_{ij}=1} p_{ij} \\
&st \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 1 \text{ for } i = s \\
&\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = -1 \text{ for } i = t \\
&\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0 \text{ for all } i \in N - \{s\} - \{t\} \\
&x_{ij} \geq 0 \text{ for all } (i,j) \in A.
\end{aligned}$$

In case you are not familiar with it, the $\prod$ symbol is just like the $\sum$ symbol, except that you *multiply* all the terms together instead of adding them. This model looks very similar to the shortest path linear programming problem, except for the objective function. However, we can easily transform this into a shortest path problem by taking a logarithm of the product of the probabilities, since for $x_{ij} \in \{0,1\}$

$$\prod_{(i,j):x_{ij}=1} p_{ij} = \exp\left(\sum_{(i,j)\in A} \ln(p_{ij}) x_{ij} \right).$$

Now, since $e^{f(x)}$ achieves its maximum exactly where $f(x)$ does, we can formulate this as a linear programming problem:

$$\begin{aligned}
&\max \sum_{(i,j)\in A} \ln(p_{ij}) x_{ij} \\
&st \sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 1 \text{ for } i = s \\
&\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = -1 \text{ for } i = t \\
&\sum_{\{j:(i,j)\in A\}} x_{ij} - \sum_{\{j:(j,i)\in A\}} x_{ji} = 0 \text{ for all } i \in N - \{s\} - \{t\} \\
&x_{ij} \geq 0 \text{ for all } (i,j) \in A.
\end{aligned}$$

But this is simply a shortest path problem. Again, we are assured that the decision variables will all be either 0 or 1, even though the value of the objective function will most likely not be an integer.

So far, we have considered problems that are “nice” in the sense that solutions are always easy to find using linear programming techniques. With maximum flow and shortest path, we never have to deal with the possibility that the optimal solution will not be integral. Unfortunately, things don’t always work out so easily. Therefore, we next present a class of problems where linear programming techniques do *not* yield integers in general. This means that we have to formulate the problem as an integer program, and that we cannot solve this type of problem “efficiently” (for a more thorough explanation of what we mean here, read chapter 1 of [4]).

The Knapsack (Rucksack) Problem

You are a soldier in a 2-1/2 ton truck bringing critical spare parts to the front lines. However, your truck hits a landmine, and is rendered inoperative. Grabbing your rucksack, you empty it of its contents and climb in the rear of the truck. You want to load your rucksack with as many spare parts as possible, and continue the mission. Because there are far too many spare parts, you must choose the parts that are the most important. Your rucksack will hold 6 cubic feet of spare

parts. You have a message from the brigade commander that lists the four most critical spare parts, their relative worth, or utility, to the accomplishment of the mission, and their volume in cubic feet. They are:

Spare Part #	Weapon system	utility	volume
1	M109A6 Paladin fuel pump	35	1
2	M1A2 Abrams optical sight	20	3
3	M2 Bradley Chain Gun bolt	30	2
4	OH-58D laser designator	55	4

We can model this situation in order to maximize utility with the following binary integer programming problem:

$$\begin{aligned} \max \quad & \sum_{j=1}^{4} u_j x_j \\ st \quad & \sum_{j=1}^{4} v_j x_j \le V \\ & x_j \in \{0,1\} \; \forall j \end{aligned}$$

Here u_i is the utility of spare part *i*, v_i is the volume of spare part *i*, V is the capacity of your rucksack, and x_j is the decision variable that equals 1 if you include spare part *i* in your rucksack and 0 if you don't.

Although this seems very similar to the models given on the previous pages, the restriction that each x_j must be either 0 or 1 makes this *type* of problem very hard to solve. You can probably solve *this example* with ease (just try all the possibilities!), but (larger) problems of this type are not solvable by any known efficient solution procedure.

The fact that finding these solutions can be difficult doesn't make these types of problems any less important, however. The Army, as well as many other organizations, needs to solve this kind of problem all the time. As a result, a great deal of research is conducted every year to attack this class of problems. This area of mathematics is called combinatorial optimization. You can learn more about it by reading [5].

Exercises

1. Write down explicitly the linear programs in the maximum flow and shortest path examples.

2. Write down the integer program given in the rucksack example.

3. What is the role of x_{ts} in the maximum flow example? Why is the associated capacity set at infinity?

4. Write an essay that compares and contrasts linear, integer, and binary integer programming.

5. Name three military problems that are maximum flow problems. What are the decision variables? What are the arc capacities?

6. Name three military problems that are shortest path problems. What are the decision variables? What are the arc lengths?

References

[1] Ahuja, Magnati, and Orlin, *Network Flows,* Prentice Hall, Englewood Cliffs, NJ (1993).

[2] Bazarra, Jarvis, and Sherali, *Linear Programming and Network Flows* (2d ed.), John Wiley & Sons, New York (1990).

[3] Chvátal, *Linear Programming*, W. H. Freeman, New York (1983).

[4] Garey and Johnson, *Computers and Intractability*, W. H. Freeman, New York (1979).

[5] Nemhauser and Wolsey, *Combinatorial Optimization*, Wiley, New York (1988).