Linear Programming Motivation

CS 149 Staff

September 11, 2007

The slides define a combinatorial optimization problem as:

> Given a set of variables, each associated with a value domain, and given constraints over the variables, find an assignment of values to variables such that the constraints are satisfied and an objective function over the variables is minimized (or maximized)!

This definition is extremely general and includes the natural expression of almost every optimization problem whose decision variant is in NP[1]. The only practical problems that come to mind that cannot be expressed naturally as combinatorial optimization problems are ones whose decision variant is not in NP[2], such as the halting problem or many problems in logic and model checking. These techniques are useful primarily for two purposes. The primary strength of combinatorial optimization techniques is to solve practical problems for which no polynomial time algorithm is known. These problems include conjunctive normal form satisfibility, traveling salesman problem, the market split problem of yesterday, sudoku, and many, many more. The second use of combinatorial optimization techniques is to solve linear programs in polynomial time. If a problem can be expressed as a linear program, it is solvable in polynomial time. When confronted with such a problem, one can try to express the problem in a standard form (such as min-cost flow) and use a specialized solver. If either the problem isn't standard or programmer time is scarcer than CPU time, then passing a linear program to a standard solver allows a painless and reasonably efficient solution.

One example problem of the linear-programming sort is the transportation problem, also known as min cost flow on a bipartite graph. Figure 1 shows an example of this problem, which is also in the slides. The vertices in this graph are retailers and sugar factories, which are labeled with demand and supply respectively. Arcs in the graph represent possible transportation routes, labeled with the cost per unit of

[1]Complexity theory deals with yes or no decision problems, not optimization problems with a multi-bit output. This means that it's technically not meaningful to say whether or not the knapsack problem is in NP. A standard abuse of notation is to say that an optimization problem to be in NP if its decision variant is. For example, is there a way to steal at least k dollars for some k, rather than how much can I steal.

[2]NP-hardness implies that any problem in NP can be expressed as a combinatorial optimization problem. The non-vacuous part of this statement is that the reduction is a simple and intuitive one. The problem of simulating a nondeterministic Turing machine is an example of a problem that is only expressible as a combinatorial optimization problem with a lot of work.

flow. The slides show a heuristic technique called matrix minimum method which will always provide a solution but not necessarily the optimal solution. This is a greedy technique, at each step moving sugar along the cheapest edge from a factory with excess supply to a retailer with excess demand. In this example, the moves are: two units from factory A to retailer 1, 2 from C to 2, 2 from A to 2, 1 from B to 2, and 6 from B to 3. The second number appearing next to the edges in the slides indicates the flow through that edge.

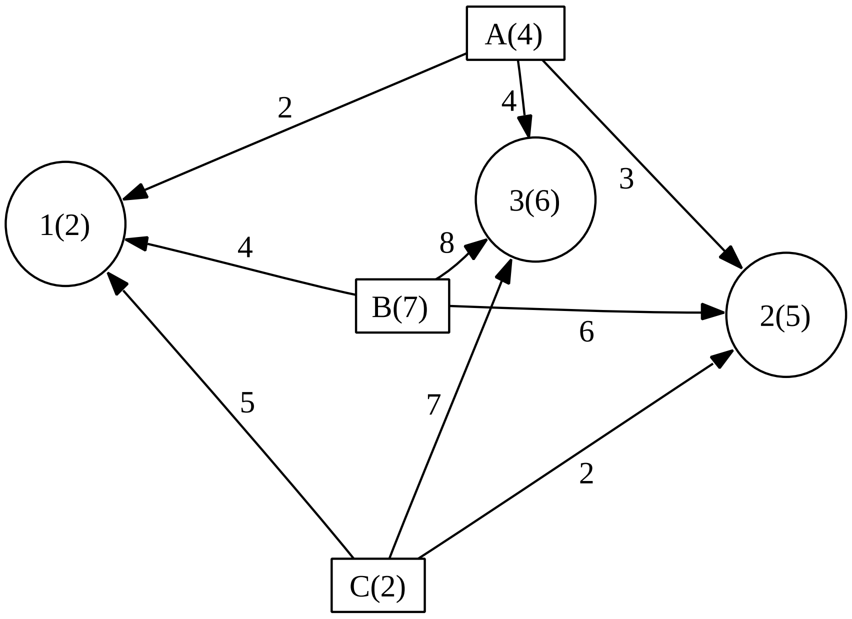


Figure 1: A transportation problem. A label on a node of form 1(2) indicates that the name of the node is 1 and the demand/supply is 2. Edge labels are costs.

The next obvious question is whether this feasible solution can be improved. To do this we need to construct the so-called residual graph. This graph intuitively shows what changes can be made and what those changes cost. The vertices of this graph are the retailers and factories as before. Each directed edge is labeled with a capacity and a cost. For edges from factories to retailers, the capacity is infinite and the cost is the same as for the original problem. There is an edge from a retailer to a factory if and only if the current solution has flow from the factory to the retailer. If the edge exists, its capacity is equal to the flow in the current solution and the cost is negative of the forward cost. The residual graph for the example problem is shown in Figure 2.

Any negative-cost cycle in this graph gives a way to modify the current solution to improve its cost. The cycle A3B1A has cost -2. We can push two units around the cycle, saving $4. After updating the residual graph for this change, there is another negative cost cycle: A3B2A of cost -1. After using that cycle the residual graph contains no negative cost cycles.

The final residual graph, shown in Figure 3, contains no negative cost cycles. One can show that for any min cost flow problem (of which this is a special case),

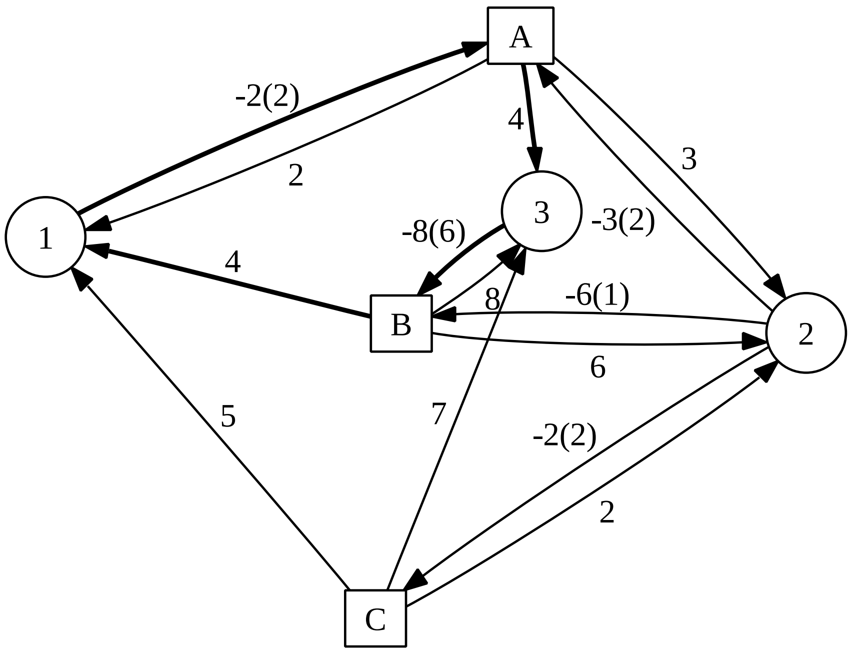


Figure 2: The initial residual graph (negative cost cycle is bold). An edge label of the form "2" indicates cost of 2 and unlimited capacity, whereas a label of the form "-3(2)" indicates a cost of -3 and capacity of 2.

a solution is optimal if and only if there are no negative cost cycles in the residual graph. (Can you prove this?) Before we generalize our insight later in the chapter, first consider another example problem.

The next example problem is the diet problem. The challenge is to determine the cheapest possible diet for your people while satisfying their dietary needs. An example instance of this problem has two foods: meat and bread. Since there are only two foods, we can construct a diagram of the feasible region as shown in the slides and in Figure 4. Each nutrient induces a linear constraint on the amount of meat and bread.

To determine the optimal point in the feasible region is helpful to find a line such that every point on that line has the same cost. We can then find the optimal point by shifting this line along its normal vector until it is tangent to the feasible region.

Yet another problem is a production planning problem where one wants to maximize profit subject to limited resources. The interesting part of this problem for our purposes is that its mathematical formulation is extremely similar to the diet problem: linear objective function and linear constraints in two variables. The same graphical technique that worked on the diet problem works on production problem as well. The slides give an example.

Problems with linear objective function and linear constraints turn up so frequently that they have a special name: linear programming (or linear optimization) problems. A fully general linear programming problem, with unrestricted (positive

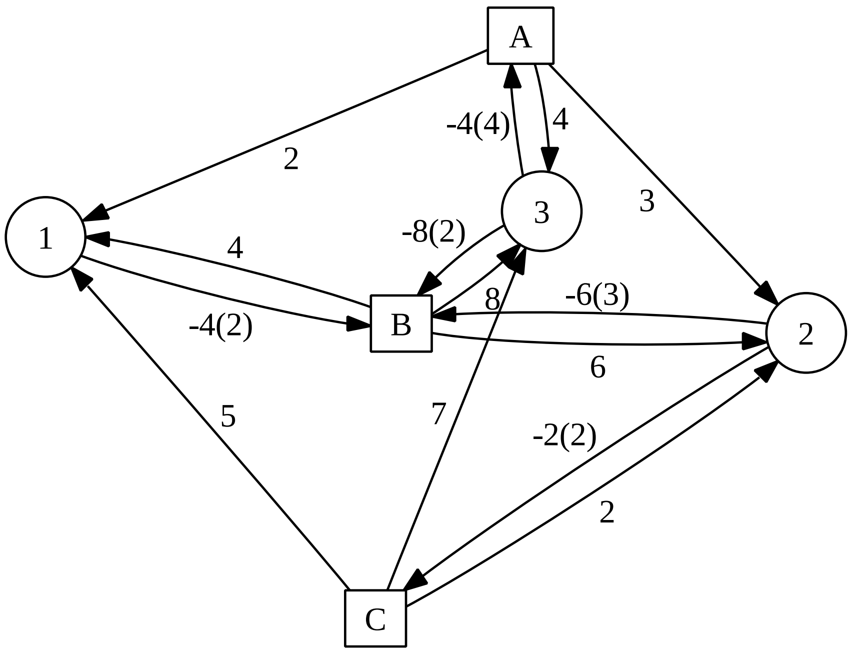
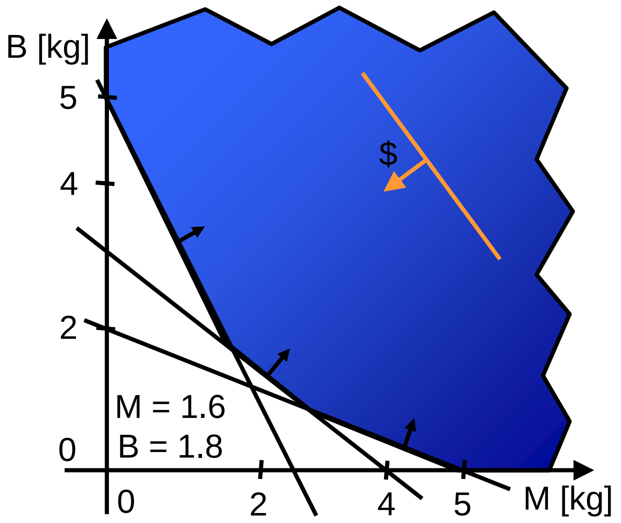


Figure 3: The final residual graph

or negative) variables, inequality constraints and equality constraints, is complicated to work with, so problems are preprocessed into one of two simpler forms. The first simplification is to insist that variables take only nonnegative values. (While intuitively this may seem I can step in the wrong direction, this actually helps for reasons that will be clear eventually.) The second simplification is to have only one type of constraints. If the problem has nonnegative variables and equality constraints, it is a standard form. If the constraints are inequality constraints, it is in canonical form.

Examples – Diet Problem



Minimize 25M + 15B

100M + 250B – 500

100M + 50B – 250

150M + 200B – 600

Figure 4: The diet problem