Homework 7, CS561, Fall 2014

Aaron Gonzales

December 2, 2014

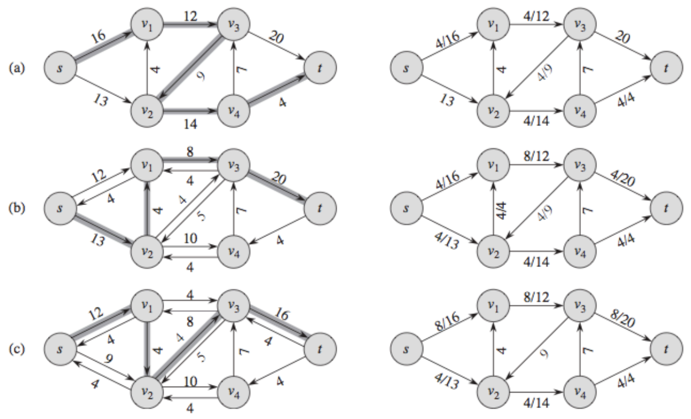

Figure 26.6 The execution of the basic Ford-Fulkerson algorithm. **(a)–(e)** Successive iterations of the **while** loop. The left side of each part shows the residual network G_f from line 3 with a shaded augmenting path p. The right side of each part shows the new flow f that results from augmenting f by f_p. The residual network in (a) is the input network G.

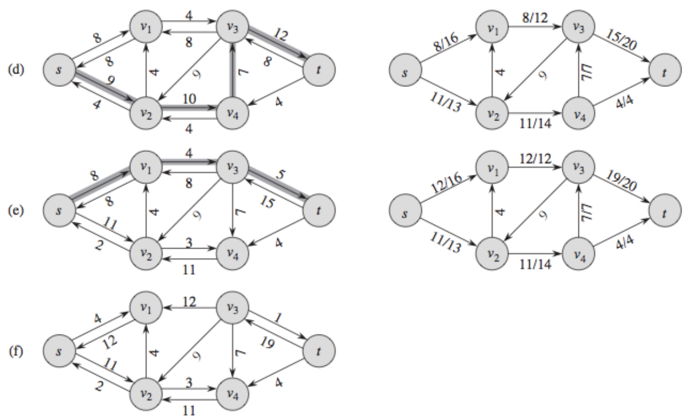

Figure 26.6, continued **(f)** The residual network at the last **while** loop test. It has no augmenting paths, and the flow f shown in (e) is therefore a maximum flow. The value of the maximum flow found is 23.

Figure 1: Figure 26.6 a-f from CLRS 3rd edition pdf

1 26.2 - 4: Min cut/Max flow

In the example of Figure 26.6, what is the minimum cut corresponding to the maximum flow shown? Of the augmenting paths appearing in the example, which one cancels flow?

Answer:

Our cut will partition the graph into two sets:

$$\{s, v_1, v_2, v_4\}, \{v_3, t\}$$

which respect the total augmenting flow in 26.6(c) of 23.

2 26.2 - 11 : connectivity

The edge connectivity of an undirected graph is the minimum number k of edges that must be removed to disconnect the graph. For example, the edge connectivity of a tree is 1, and the edge connectivity of a cyclic chain of vertices is 2. Show how to determine the edge connectivity of an undirected graph $G = (V, E)$ by running a maximum-flow algorithm on at most $|V|$ flow networks, each having $O(V)$ vertices and $O(E)$ edges.

Answer:

As a reminder, a min cut can be stated as finding a subset $S \in V$ of a graph $G = (V, E)$such that $S \neq 0$ with edges C crossing from A to $V - A$ minimized. For this problem, the cost of the cut can be the number of edges $||C||$ that cross the cut. Here, we want to find the min cut of some cost equal to k. Removing the edge set C from the graph clearly disconnects the graph.

Our algorithm can be described as follows: Let s be some vertex in V chosen at random. give a weight of 1 to every edge in the graph. For every other vertex t_i in G, find the min cut from s to t_i and save the number of edges that cross the min cut. Upon completion, return minimum number k from the process and you have the edge connectivity value of the graph. The algorithm examines $\Theta(V - 1) = \Theta(V)$ nodes and using Ford-Fulkerson gives us a total runnig time of $O(E \times V^2)$.

Note that Karger's algorithm can answer this question as well in a randomized way of contracting edges in a graph by merging nodes together into a series of multigraphs until two nodes remain that represent a cut in the original graph. It can run with total time $O(V^2 \, log \, V)$ depending on implementation and would work far more quickly than the above answer[1].

3 Perfect Matchings

A bipartite graph is a graph that contains no cycle with an odd number of edges. Recall from the wrestler problem that a graph, $G = (V, E)$ is bipartite iff V can be partitioned into two sets L, R such that all edges in E have one endpoint in L and one endpoint in R. A matching in a graph is a set of edges $E' \in E$ such that each vertex in V is matched at most once, i.e. it is incident to a most one edge in E'. A perfect matching is a matching where every vertex is matched, i.e. is incident to exactly one edge in E'. For a set of vertices $S \in V$, let $N(S)$ be the set of all neighbors of S, i.e. $\{y \in V : (x, y) \in E$ for some $x \in S\}$
Assume we are given a bipartite graph where $|L| = |R|$. Prove that there is a perfect matching in G iff $|N(S)| \geq |S|$ for all $S \in L$. Hint: Use the Max flow/Min Cut Theorem

Answer:

If we let assign unit (1) weight to each edge in the graph and add a source vertex s that is connected from itself to each vertex in L and a sink vertex t that takes a directed edge from each vertex in R, we can model this as a network flow problem. (In other words, direct all the edges from the source on the left to the sink of the right). Let this restructured graph be called G'.

Proof. Given that our graph has $|L| = |R|$, let $n = |L|$. Suppose for sake of contridiction that G contains no perfect matching. If this is true, then the size of the maximum match in G is at most $\leq n - 1$ and as such the max flow in G' is $\leq n - 1$. As such, G' must have a cut C with at least capacity $n - 1$.

Let L_a be the left vertices in cut C, L_b be the remaining vertices in L. Let R_a and R_b be defined the same for the right side. The flow capacity of the edges crossing C is the same as the number of edges crossing C, which is $|L_b| + |R_a| + |L_a, R_b|$. We now have

$$n - 1 \geq |L_b| + |R_a| + |L_1, R_2|$$

and this reduces to

$$|N(L_a) \leq |R_a| + |L_a, R_2|$$

[1]http://en.wikipedia.org/wiki/Karger%27s_algorithm

because our L_a neighboorhood can only have edges from L_a, R_b in R_2.

$$L_a \leq N(L_a) + 1$$

and we conclude the proof. □

4 Image segmentation

Consider the following problem related to segmenting the pixels of an image between foreground and background. We have a picture that consists of n pixels. We represent this as an undirected graph $G = (V, E)$ where V is the set of pixels and there is an edge $(i, j) \in E$ iff pixel i and pixel j are neighbors in the image [2].
We want to find a good segmentation, which is an assignment of each pixel to either the foreground or the background. For each pixel i, we have a likelihood a_i that i belongs to the foreground and a likelihood b_i that i belongs to the background. These likelihood values are all non-negative. Additionally, for each edge $(i, j) \in E$, we have a non-negative separation penalty p_i, j which is charged if one of i or j is assigned to the foreground and the other is assigned to the background. Our problem then is to find a partition of the set of pixels into sets A and B so as to maximize:

$$L(A, B) = \sum_{i \in A} a_i + \sum_{i \in B} b_i \sum_{(i,j) \in E, |A \cap \{i,j\}| = 1} p_{i,j} \tag{1}$$

Give an efficient algorithm to solve this problem

Answer:

This maximization problem stated in 1 can be formulated as a minimization problem instead, that is,

$$\max(a) = \sum_{i \in A} a_i + \sum_{i \in B} b_i \sum_{(i,j) \in E, |A \cap \{i,j\}| = 1} p_{i,j} \tag{2}$$

is equivalent to

$$\min(g') = \sum_{i \in B} f_i + \sum_{i \in A} b_i + \sum_{i \in P | j \in Q j \in P | i \in Q} p_{ij}. \tag{3}$$

The above minimization problem can be formulated as a minimum-cut problem by constructing a network where the source is connected to all the pixels with capacity a_i and the sink is connected by all the pixels with capacity b_i Two directed edges i, j and j, i with p_{ij} capacity are added between two adjacent pixels. The s-t cut-set then represents the pixels assigned to the foreground in A and pixels assigned to background in B as we have minimized our desired quantity from equation 3.

5 Exercise 29.2-2 (Linear Program)

Write out explicitly the linear program corresponding to finding the shortest path from node s to node y in Figure 24.2(a).

[2] Note that we'd commonly expect this graph to be a grid, but in fact we want to handle any arbitrary graph (to handle, e.g., 3-D images, wrapped and warped images, etc)

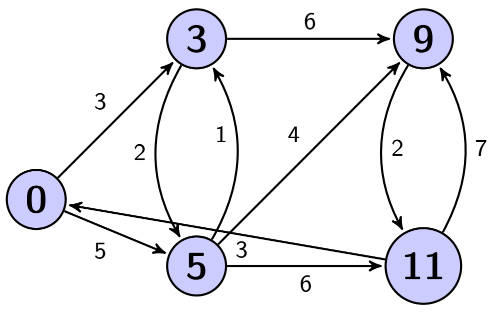

Figure 2: Figure 24.2 from CLRS

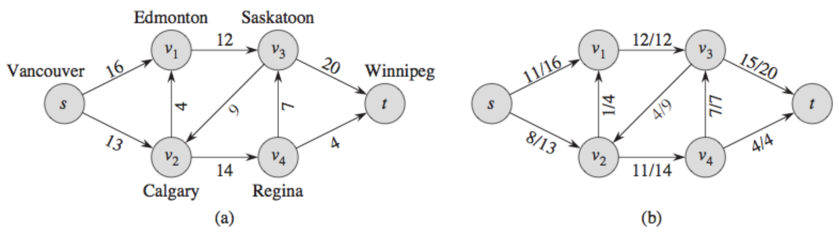

Figure 26.1 **(a)** A flow network $G = (V, E)$ for the Lucky Puck Company's trucking problem. The Vancouver factory is the source s, and the Winnipeg warehouse is the sink t. The company ships pucks through intermediate cities, but only $c(u, v)$ crates per day can go from city u to city v. Each edge is labeled with its capacity. **(b)** A flow f in G with value $|f| = 19$. Each edge (u, v) is labeled by $f(u, v)/c(u, v)$. The slash notation merely separates the flow and capacity; it does not indicate division.

Figure 3: Figure 26.1a from CLRS 3rd edition PDF

Answer:

We have several constraints for our linear program. Let d_a be the distance up to a node a. Our constraints follow:

$$\begin{aligned}
d_s &= 0 \\
d_t &\le d_s + 3 \\
d_t &\le d_y + 1 \\
d_y &\le d_s + 5 \\
d_y &\le d_t + 2 \\
d_x &\le d_t + 6 \\
d_x &\le d_z + 7 \\
d_z &\le d_y + 6 \\
d_z &\le d_x + 2
\end{aligned}$$

6 Exercise 29.2-4 (Network Flow as an LP)

Write out explicitly the linear program corresponding to finding the maximum flow in Figure 26.1(a) (Figure 3 on page 4.

Answer:

Defining flow as a linear program can be done as such. Let $f u \to v$ be a path from node u to node v. Note that we must preserve conservation of flow and that our program will have constraints respecting flow and capacity of edges. The problem then is

$$
\begin{aligned}
\text{maximize} \sum_{v:(s,v)\in E} f_{s\to v} && (4)\\
\text{subject to} && (5)\\
\sum u : (u,v) \in E f_{u\to v} = \sum_{w:(v,w)\in E} f_{v\to w} \forall v \in V - \{s,t\} && (6)\\
f_{u\to v} \leq c(u,v) \forall (u,v) \in E && (7)\\
f_{u\to v} \geq 0 \forall (u,v) \in E && (8)
\end{aligned}
$$

Our program would have the following constraints:

$$
\begin{aligned}
f_{s\to s} &= 0\\
f_{s\to v_1} &\leq 16\\
f_{s\to v_2} &\leq 13\\
f_{v_1\to v_3} &\leq 12\\
f_{v_2\to v_1} &\leq 4\\
f_{v_2\to v_4} &\leq 14\\
f_{v_3\to v_2} &\leq 9\\
f_{v_3\to t} &\leq 20\\
f_{v_4\to v_3} &\leq 7\\
f_{v_4\to t} &\leq 4
\end{aligned} \tag{9}
$$

we also have the following constraints on capacity:

$$
\begin{aligned}
f_{s\to v_1} + f_{v_2\to v_1} &= f_{v_1\to v_3}\\
f_{s\to s} + f_{s\to s} &= f_{s\to s}\\
f_{s\to s} + f_{s\to s} &= f_{s\to s}\\
f_{s\to s} + f_{s\to s} &= f_{s\to s}\\
f_{s\to s} + f_{s\to s} &= f_{s\to s}\\
f_{s\to s} + f_{s\to s} &= f_{s\to s}\\
f_{s\to s} + f_{s\to s} &= f_{s\to s}\\
f_{s\to s} + f_{s\to s} &= f_{s\to s}\\
f_{s\to s} + f_{s\to s} &= f_{s\to s}
\end{aligned} \tag{10}
$$

7 Rock, Paper, Scissors

Rock, Paper, Scissors is a simple 2 person game. In a given round, both players simultaneously choose either Rock, Paper or Scissors. If they both choose the same object, its a tie. Otherwise, Rock beats Scissors; Scissors beats Paper; and Paper beats Rock. Imagine youre playing the following betting variant of this game with a friend. When Scissors beats Paper, or Paper beats Rock, the loser gives the winner \$1. However, in the case when Rock beats Scissors, this is called a smash, and the loser must give the winner \$10.

(a) Say you know that your friend will choose Rock, Scissors or Paper, each with probability $\frac{1}{3}$. Write a linear program to calculate the probabilities you should use of choosing each object in order to maximize your expected winnings. Let p_1, p_2, p_3 be variables associated with the best way of choosing Rock, Scissors and Paper respectively. Note: If you want to check your work, there are several free linear program solvers on the Internets: check the Wikipedia page on linear programming.

Answer:

Let $\rho = \{p_r, p_s, p_p\}$ be the set of variables associated with us choosing Rock, Scissors and Paper respectively. Maximize: if p_1, p_2, p_3 are the probabilities of choosing the

$$\frac{1}{3}\left(10 * p_r - 1 * p_s - 10 * p_s + 1 * p_s + 1 * p_p - 1 * p_p\right) \tag{11}$$

$$\text{with the following constraints:} \tag{12}$$

$$0 \le p_r, p_s, p_p \le 1 \tag{13}$$

$$\sum \rho = 1 \text{ (abuse of notation, sorry)} \tag{14}$$

(b) Now say that your friend is smart and, also, clairvoyant: she will magically know the exact probabilities you are using and will respond optimally. Write another linear program to calculate the probabilities you should now use in order to maximize your expected winnings.
Hint 1: If your opponent knows your strategy, her strategy will be to choose one of the three objects with probability 1.
Hint 2: Review the LP you wrote for the shortest paths problem.

Answer:

Let P = our profit and Q = our friend's profit. Let $\rho = \{P_r, P_p, P_s\}$ be our probabilities of choosing rock, paper, or scissors. Let Q_r, Q_p, Q_s be our friend's winnings when she chooses rock, paper, or scissors, respectively. Then let her winnings be defined as

$$\begin{aligned} Q_r &= P_p - 10P_s && \text{she picks rock} \\ Q_p &= P_s - P_r && \text{she picks paper} \\ Q_s &= 10P_r - P_p && \text{she picks scissors} \end{aligned} \tag{15}$$

Then we want to maximize her winnings in 10 in such a way to minimize our winnings.

$$\begin{aligned} &\max(Q) = \\ &\qquad \min \begin{cases} Q_r = & P_p - 10P_s \\ Q_p = & P_s - P_r \\ Q_s = & 10P_r - P_p \end{cases} \\ &0 \le p_1, p_2, p_3 \le 1 \\ &\qquad \sum \rho = 1 \text{ (abuse of notation, sorry)} \end{aligned} \tag{16}$$

8 Independent-Set

The problem INDEPENDENT-SET asks: Does there exist a set of k vertices in a graph G with no edges between them? Show that this problem is NP-Complete. (hint: Reduce from CLIQUE)

Answer:

Proof. Via reduction from CLIQUE

Part one:NP

INDEPENDENT-SET is in NP. Given a vertex set S returned from the algorithm we can indeed check that S is an independent set of vertices by checking that each vertex in S is in the graph and that there are no edges between any two vertices in S. A simple BFS or DFS will handle this in $O(V + E)$ time and INDEPENDENT-SET $\in$ NP.

Part two: NP-Hard

INDEPENDENT-SET is NP-Hard. Recall that an instance of CLIQUE is a graph G and an integer k. We can convert an instance of CLIQUE to INDEPENDENT-SET as such: Let G_c be the complement graph of G and pass G_c, k to INDEPENDENT-SET. Assume that there is a clique C of size $k \in G$. For any $u, v \in C, (u, v) \in E$. Thus, $(u, v) \notin E_c$ and the vertices of C form an independent set in G_c and G_c is a yes instance from INDEPENDENT-SET. If G_c is a yes isntance from INDEPENDENT-SET, then any $u, v \in I, (u, v,) \notin E_c$ and $u, v \in E$ and the vertices in I form a clique in G.
INDEPENDENT-SET is both NP and NP-HARD and as such IS $\in$ NP-Complete. □

9 Exercise 34.5-1 (Subgraph Isomorphism)

The subgraph-isomorphism problem takes two undirected graphs G_1 and G_2, and it asks whether G_1 is isomorphic to a subgraph of G_2. Show that the subgraph-isomorphism problem is NP-complete.

Answer:

In order to show NP-completeness, we must state the problem as a decision problem. We can ask if a given graph is isomorphic to a subgraph of another graph, and in this case, if G_1 is isomorphic to a subgraph of G_2, the answer is "true" and "false" otherwise. Let it be stated that to be isomorphic to another graph, a graph $G = (V, E)$ must have a subgraph $G_0 = (V_0, E_0) : V_0 \subseteq V, E_0 = E \cap (V_0 \times V_0)$ such that $G_0 \cong H$? Does a mapping $f : V_0 \to V'$ exist such that vertices $v_1, v_2 \in E_0 \Leftrightarrow (f(v_1), f(v_2)) \in E'$?

Proof. **SI is in NP**

there can be $O(n^2)$ pairs in the list of nodes comprising G_1 and the subgraph of G_2 that we must check to verify that it is isomorphic. This polynomial time check verifies that SI is in NP.

SI is NP-Complete

By reduction from CLIQUE.
CLIQUE $\leq_P$ SUBGRAPH ISOMORPHISM. Let $(G = (V, E)k)$ be an input for CLIQUE and let G_1 to be the complete graph on k vertices and G_2 to be the graph G. $G_1, G_2 \in$ SUBGRAPH ISOMORPHISM iff $G, k \in CLIQUE$. If G has a k-clique, then G has the k-graph as a subgraph and G, G_k is a yes instance of the problem. As this is at least as hard as Clique, we conclude that SUBGRAPH-ISOMORPHISM is NP-Hard $\implies$ NP-Complete. □