Advanced Algorithms Course. Lecture Notes. Part 6

A Simplified Airline Scheduling Problem

An airline has to operate m flights, each of which is characterized by origin and destination airport, departure and arrival time. For any two flights i and j we define, by ad-hoc criteria, whether flight j is *reachable* from flight i (for example: same airport, enough time between arrival of i and departure of j). This reachability relation defines a directed acyclic graph (DAG) on the set of flights. If j is reachable from i, then both flights may be performed by the same plane. The problem is: Given a directed graph describing the reachability relation, and a positive integer k, can all flights be operated using a fleet of at most k planes?

We will reduuce this problem to the Circulation problem. It is tempting to represent the airports by nodes and the flights by edges, which seems natural. However, the construction presented below is different. Note that a graph is an abstract structure. Nodes and edges do not have to correspond to any physical objects that resemble points and lines. We will not even explicitly represent airports, instead, we will use directed edges to model both the flights and their reachability relationships.

Now a possible construction follows. The pairs of numbers appearing below are lower and upper bounds on capacities (l_e, c_e).

We represent all flights by mutually disjoint directed edges with capacities $(1, 1)$. The lower capacity is 1, since every flight must be performed. Whenever a flight g is reachable from another flight f, we insert a directed edge from the end of f to to the start of g, with capacity $(0, 1)$. Here the lower capacity is 0, since we may or may not use this connection. Furthermore, we insert a source s and a sink t. We connect s to the start of every flight, by an edge of capacity $(0, 1)$. Similarly, we connect the end of every flight to t, by an edge of capacity $(0, 1)$. Finally we connect s to t by an edge

of capacity $(0, k)$. Nodes s and t have demands $-k$ and $+k$, respectively, and all other nodes have demand 0.

In this network, the possible schedules (assignments of planes to flights) correspond to the possible circulations. Every flow unit models one plane, and the flow on the extra edge from s directly to t is the number of unused planes.

The equivalence proof requires a bit of care and essentially uses the fact that the reachability graph is a DAG. However, we omit the details. And once again we stress the usefulness of the lower capacity bounds.

A Machine Learning Problem: Image Segmentation

In this problem, our aim is to label every pixel of a digital image as foreground (part of an object) or background. The image is represented as an undirected graph $G = (V, E)$ where nodes are pixels, and edges exist between any two neighbored pixels (according to some definition of neighborhood). For example, the graph can simply be a grid where every pixel has an edge to its four neighbors in the four cardinal directions.

For every pixel i we are also given two numbers a_i and b_i expressing the strength of belief that pixel i is foreground or background, respectively. We do not discuss here in depth how these values are obtained. Criteria could be, for example, the colors and positions of pixels.

A further assumption is that the picture does not comprise too many switches between foreground and background, that is, it shows a few large and connected objects with rather smooth shapes. Therefore we introduce penalties for label switches: For each pair of neighbored pixels i, j we charge a penalty p_{ij} if i and j have different labels.

Altogether this gives rise to the following optimization problem: Split V into sets A and B (foreground and background) so as to maximize

$$q(A, B) := \sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E, i \in A, j \in B} p_{ij}.$$

That is, the segmentation should respect the classification criteria for the single pixels, but at the same time it should not need too many switches.

We emphasize that the model assumptions and the purely mathematical optimization problem derived from them are two separate aspects, and we will consider the latter aspect only.

We can reduce this problem to Minimum Cut as follows. First observe that the problem is equivalent to minimizing

$$q'(A,B) := \sum_{i \in A} b_i + \sum_{j \in B} a_j + \sum_{(i,j) \in E, i \in A, j \in B} p_{ij}.$$

That is, we want to minimize the penalties for false labels and for switches. As we need a directed graph, we replace every edge (i,j) with two opposite directed edges having the same capacity p_{ij}. We insert a source s and a sink t, and for every pixel k we insert a directed edges from s to k with capacity a_k, and from k to t with capacity b_k. Now any $s-t$ cut (A,B) has capacity $q'(A,B)$. (For seeing this, a drawing of a network with a few pixels may be helpful.) Thus, an optimal segmentation corresponds to a minimum cut, and by computing a minimum cut we can solve our problem.

Project Selection

Let P be a set of possible projects to choose from. Project i has revenue p_i. A value p_i can also be negative, in which case the project is an investment for other projects: Some projects depend on others, and these dependencies are given as a directed graph $G = (P, E)$ where an edge (i,j) means: if i shall be done, then j must be done, too (before i can even start). Clearly, G must be a DAG, since projects in a directed cycle of dependencies can never be done. We call a set of projects $A \subset P$ feasible if A respects these precedence constraints. The problem is to select a feasible set A that maximizes $\sum_{i \in A} p_i$. This is also known as the Open-Pit Mining problem; one can easily imagine the reason.

A remark on the notation: Edges are directed here from i to j, if j must be done before i. That is, the "arrow" is opposite to the chronological order. This is not a mistake, but only a convention; you may reverse all edges if you like. Or read the arrows as "if i then j" ($i \Longrightarrow j$).

We reduce Project Selection (Open-Pit Mining) to Minimum Cut as follows. We insert a source s and a sink t. The directed edges are (s,i) with capacity p_i, if $p_i > 0$, and (i,t) with capacity $-p_i$, if $p_i < 0$. The nice trick is to give the directed edges in G (which indicate the precedence constraints) a huge capacity. This has the effect that none of these directed edges can go from A to B in a minimum cut $(A \cup \{s\}, B \cup \{t\})$. Hence A is automatically a feasible solution whenever $(A \cup \{s\}, B \cup \{t\})$ is a minimum cut. Now we can solve the Minimum Cut problem in this network, without worrying about the feasibilty of A.

It remains to show that minimizing the cut capacity is in fact equivalent to maximizing the revenue. This is proved in a few lines:

$$c(A \cup \{s\}, B \cup \{t\}) = \sum_{p_i>0, i\in B} p_i - \sum_{p_i<0, i\in A} p_i$$

holds by the definition of capacity. We artificially add zero:

$$c(A \cup \{s\}, B \cup \{t\}) = \sum_{p_i>0, i\in B} p_i - \sum_{p_i<0, i\in A} p_i - \sum_{p_i>0, i\in A} p_i + \sum_{p_i>0, i\in A} p_i.$$

Now we can group the terms in a different way:

$$c(A \cup \{s\}, B \cup \{t\}) = \sum_{p_i>0} p_i - \sum_{i\in A} p_i.$$

Note that the first term is constant and the second term is the revenue.