Chapter 10

Matchings, Cuts, and Flows

A joyful life is an individual creation that cannot be copied from a recipe.

—Mihaly Csikszentmihalyi, Flow: The Psychology of Optimal Experience

While the previous chapter gave you several algorithms for a single problem, this chapter describes a single algorithm with many variations and applications. The core problem is that of finding maximum flow in a network, and the main solution strategy I’ll be using is the augmenting path method of Ford and Fulkerson. Before tackling the full problem, I’ll guide you through two simpler problems, which are basically special cases (they’re easily reduced to maximum flow). These problems, bipartite matching and disjoint paths, have many applications themselves and can be solved by more specialized algorithms. You’ll also see that the max-flow problem has a dual, the min-cut problem, which means that you’ll automatically solve both problems at the same time. The min-cut problem has several interesting applications that seem very different from those of max-flow, even if they are really closely related. Finally, I’ll give you some pointers on one way of extending the max-flow problem, by adding costs, and looking for the cheapest of the maximum flows, paving the way for applications such as min-cost bipartite matching.

The max-flow problem and its variations have almost endless applications. Douglas B. West, in his book Graph Theory (see References, Chapter 2), gives some rather obvious ones, such as determining the total capacities of road and communication networks, or even working with currents in electrical circuits. Kleinberg and Tardos (see “References” in Chapter 1) explain how to apply the formalism to survey design, airline scheduling, image segmentation, project selection, baseball elimination, and assigning doctors to holidays. Ahuja, Magnanti, and Orlin have written one of the most thorough books on the subject and cover well over a hundred applications in such diverse areas as engineering, manufacturing, scheduling, management, medicine, defense, communication, public policy, mathematics, and transportation. Although the algorithms apply to graphs, these application need not be all that graph-like at all. For example, who’d think of image segmentation as a graph problem? I’ll walk you through some of these applications in the unsurprisingly named section “Some Applications” later in the chapter. If you’re curious about how the techniques can be used, you might want to take a quick glance at that section before reading on.

The general idea that runs through this chapter is that we’re trying to get the most out of a network, moving from one side to the other, pushing through as much of we can of some kind of substance—be it edges of a bipartite matching, edge-disjoint paths, or units of flow. This is a bit different from the cautious graph exploration in the previous chapter. The basic approach of incremental improvement is still here, though. We repeatedly find ways of improving our solutions slightly, until it can’t get any better. You’ll see that the idea of canceling is key—that we may need to remove parts of a previous solution in order to make it better overall.
Note I’m using the labeling approach due to Ford and Fulkerson for the implementations in this chapter. Another perspective on the search for augmenting paths is that we’re traversing a residual network. This idea is explained in the sidebar “Residual Networks” later in the chapter.

Bipartite Matching

I’ve already exposed you to the idea of bipartite matching, both in the form of the grumpy moviegoers in Chapter 4 and the stable marriage problem in Chapter 7. In general, a matching for a graph is a node-disjoint subset of the edges. That is, we select some of the edges in such a way that no two edges share a node. This means that each edge matches two pairs—hence the name. A special kind of matching applies to bipartite graphs, graphs that can be partitioned into two independent node sets (subgraphs without edges), such as the graph in Figure 10-1. This is exactly the kind of matching we’ve been working with in the moviegoer and marriage problems, and it’s much easier to deal with than the general kind. When we talk about bipartite matching, we usually want a maximum matching, one that consists of a maximum number of edges. That means that, if possible, we’d like a perfect matching, one where all nodes are matched. This is a simple problem, but one that can easily occur in real life. Let’s say, for example, you’re assigning people to projects, and the graph represents who’d like to work on what. A perfect matching would please everyone.1

Figure 10-1. A bipartite graph with a (non-maximal) matching (heavy edges) and an augmenting path from b to f (highlighted)

1 If you allow them to specify a degree of preference, this turns into the more general min-cost bipartite matching, or the assignment problem. Although a highly useful problem, it’s a bit harder to solve—I’ll get to that later.