Advanced Algorithms

Class Notes for Thursday, October 26, 2012

Bernard Moret

1 Iterative Improvement Algorithms

1.1 Bipartite matching: algorithms

We saw on Monday that we can produce an optimal solution (a maximum matching) by using augmenting paths. The question now is how to implement this general approach and to analyze the resulting algorithm(s). The general approach is simply

- Begin with an arbitrary (possibly empty) matching.
- Repeatedly discover an augmenting path and switch the status of all edges along it, until no augmenting path can be found.

The second step cannot be repeated more than a linear number of times. Maximum matching thus reduces to the problem of discovering augmenting paths.

In a bipartite graph, any augmenting path begins on one side of the graph and ends on the other. Thus a search algorithm can simply start at any unmatched vertex on one side of the graph, say the left side, and traverse any edge to the other side. If the endpoint on the right side is also unmatched, then an augmenting path, consisting of a single unmatched edge, has been found. If the other endpoint is matched, then the algorithm traverses that matched edge to the left side and follows any unmatched edge, if one exists, to an unvisited vertex on the right side. The process is repeated until either an augmenting path is found or a dead end on the left side is reached. Unmatched edges are always traversed from the left side to the right side and matched edges in the opposite direction. If a dead end is reached, we must explore other paths until we find an augmenting path or run out of possibilities. In developing an augmenting path, choices arise in only two places: in selecting an initial unmatched vertex and in selecting an unmatched edge out of a vertex on the left side. In order to examine all possibilities for augmenting paths, we need to explore these choices in some systematic way; because all augmenting paths make exactly the same contribution of one additional matched edge, we should search for the shortest augmenting paths.

Thus we use a breadth-first search of the graph, starting at each unmatched vertex on the left side. If any of the current active vertices (the frontier in the BFS, which will always be vertices on the left side) has an unmatched neighbor on the right, we are done. Otherwise, from each neighbor on the right, we follow the matched edge of which it is an endpoint back to a vertex on the left and repeat the process. Thus the BFS increases path lengths by 2 at each iteration—because the move back to the left along matched edges is forced. The BFS takes $O(|E|)$ time, as it cannot look at an edge more than twice (once from each end); as the number of augmenting paths we may find is in $O(|V|)$, the running

time of this BFS augmenting strategy is $O(|V| \cdot |E|)$. Since the input size is $\Theta(|V| + |E|)$, the time taken is more than linear, but no more than quadratic, in the size of the input.

However, we are wasting a lot of time: each new BFS starts from scratch and, most likely, will follow many paths already followed in the previous BFS. And each BFS produces a single new augmenting path. Yet, there typically will be a number of augmenting paths in a graph with respect to a matching, especially if that matching is small. Instead of stopping at the first unmatched neighbor on the right, we could finish that stage of BFS, collecting all unmatched neighbors on the right. Doing so would not increase the worst-case running time of the BFS, yet might yield multiple augmenting paths of the same length. However, we can use multiple augmenting paths only if they are vertex-disjoint, since otherwise we could cause conflicting assignments of vertices or even edges. The BFS might discover that k_l of the current active vertices (on the left) have an unmatched neighbor on the right, but some of these neighbors might be shared; if there are k_r unmatched neighbors on the right, the maximum number of disjoint augmenting paths is $\min\{k_l, k_r\}$. The number may be smaller, however, because this sharing of vertices can occur at any stage along the alternating paths. Thus we must adjust our BFS to provide backpointers, so that we can retrace paths from right-side unmatched vertices reached in the search; and we must add a backtracing phase, which retraces at most one path for each unmatched vertex reached on the right-hand side. The backtracing is itself a graph search. Specifically, for each left-side vertex encountered during the breadth-first search, we record its distance from the closest unmatched left-side vertex, passing as before through matched right-side vertices. We use this information to run a (backward) depth-first search from each unmatched right-side vertex discovered during the BFS: during a DFS we consider only edges that take us one level closer to unmatched left-side vertices. When we discover an augmenting path, we eliminate the vertices along this path from consideration by any remaining DFS, thereby ensuring that our augmenting paths will be vertex-disjoint.

We can hope that the number of augmenting paths found during each search is more than a constant, so that the number of searches (iterations) to be run is significantly decreased, preferably to $o(|V|)$. We characterize the gain to be realized through a series of small theorems; as in the previous lecture, these theorems apply equally to general graphs and bipartite graphs. We begin with a more precise proof of Berge's theorem that allows us to refine its conclusion.

Theorem 1. *Let M_1 and M_2 be two matchings in some graph, $G = (V, E)$, with $|M_1| > |M_2|$. Then the subgraph $G' = (V, M_1 \oplus M_2)$ contains at least $|M_1| - |M_2|$ vertex-disjoint augmenting paths with respect to M_2.*

Proof. Recall that every connected component of G' is one of: (i) a single vertex; (ii) a cycle of even length, with edges alternately drawn from M_1 and M_2; or (iii) a path with edges alternately drawn from M_1 and M_2. Let $C_i = (V_i, E_i)$ be the ith connected component and define $\delta(C_i) = |E_i \cap M_1| - |E_i \cap M_2|$. From our previous observations, we know that $\delta(C_i)$ must be one of -1, 0, or 1 and that it equals 1 exactly when C_i is an augmenting path

with respect to M_2. Now we have

$$\sum_i \delta(C_i) = |M_1 - M_2| - |M_2 - M_1| = |M_1| - |M_2|,$$

so that at least $|M_1| - |M_2|$ components C_i are such that $\delta(C_i)$ equals 1, which proves the theorem. □

This tells us that many disjoint augmenting paths exist, but says nothing about their lengths, nor about finding them. Indeed, if we take M_2 to be the empty set and M_1 to be a maximum matching, the theorem tells us that the original graph contains enough disjoint augmenting paths to go from no matching at all to a maximum matching in a single step! But these paths will normally be of various lengths and finding such a set is actually a very hard problem. We will focus on finding a set of disjoint *shortest* augmenting paths (thus all of the same length) with respect to the current matching; such a set will normally not contain enough paths to obtain a maximum matching in one step.

Our next result is intuitively obvious, but the theorem proves it and also makes it precise: successive shortest augmenting paths cannot become shorter.

Theorem 2. *Let $G = (V, E)$ be a graph, with M a nonmaximal matching, P a shortest augmenting path with respect to M, and P' any augmenting path with respect to the augmented matching $M \oplus P$. Then we have*

$$|P'| \geq |P| + |P \cap P'|$$

Proof. The matching $M \oplus P \oplus P'$ contains two more edges than M, so that, by our previous theorem, $M \oplus (M \oplus P \oplus P') = P \oplus P'$ contains (at least) two vertex-disjoint augmenting paths with respect to M, call them P_1 and P_2. Thus we have $|P \oplus P'| \geq |P_1| + |P_2|$. Since P is a shortest augmenting path with respect to M, we also have $|P| \leq |P_1|$ and $|P| \leq |P_2|$, so that we get $|P \oplus P'| \geq 2|P|$. Since $P \oplus P'$ is $(P \cup P') - (P \cap P')$, we can write $|P \oplus P'| = |P| + |P'| - |P \cap P'|$. Substituting in our preceding inequality yields our conclusion. □

An interesting corollary (especially given our BFS approach to finding shortest augmenting paths) is that two successive shortest augmenting paths have the same length only if they are disjoint. Our new algorithm uses all disjoint shortest augmenting paths it finds, as follows.

- Begin with an arbitrary (possibly empty) matching.
- Repeatedly find a maximal set of vertex-disjoint shortest augmenting paths, and use them all to augment the current matching, until no augmenting path can be found.

Now we are ready to prove the crucial result on the worst-case number of searches required to obtain a maximum matching. The result itself is on the number of different lengths that can be found among the collection of shortest augmenting paths produced in successive searches.

Theorem 3. *Let s be the cardinality of a maximum matching and let $P_1, P_2, \ldots, P_s$ be a sequence of shortest augmenting paths that build on the empty matching. Then the number of distinct integers in the sequence $|P_1|, |P_2|, \ldots, |P_s|$ cannot exceed $2\lfloor\sqrt{s}\rfloor$.*

The intuition here is that, as we start the first search with an empty matching (or a small one), there will be many disjoint shortest augmenting paths and so there will be many repeated values towards the beginning of the sequence of path lengths; toward the end, however, augmenting paths are more complex, longer, and rarer, so that most values toward the end of the sequence will be distinct. The proof formalizes this intuition by using a "midpoint" in the number of distinct values that is very far along the sequence of augmenting paths: not at $\frac{s}{2}$, but at $s - \lfloor\sqrt{s}\rfloor$.

Proof. Let $r = \lfloor s - \sqrt{s}\rfloor$ and consider M_r, the rth matching in the augmentation sequence. Since $|M_r| = r$ and since the maximum matching has cardinality $s > r$, we conclude (using Berge's extended theorem) that there exist exactly $s - r$ vertex-disjoint augmenting paths with respect to M_r. (These need not be the remaining augmenting paths in our sequence, P_{r+1}, P_{r+2}, …, P_s.) Altogether these paths contain at most all of the edges from M_r, so that the shortest contains at most $\lfloor r/(s-r)\rfloor$ such edges (if the edges of M_r are evenly distributed among the $s - r$ vertex-disjoint paths) and thus at most $2\lfloor r/(s-r)\rfloor + 1$ edges in all. But the shortest augmenting path is precisely the next one picked, so that we get

$$\begin{aligned} |P_{r+1}| &\leq 2\left\lfloor \lfloor s-\sqrt{s}\rfloor / (s - \lfloor s - \ sqrts \rfloor)\right\rfloor + 1 \\ &\leq 2(s-\sqrt{s})/\sqrt{s} + 1 \\ &\leq 2\sqrt{s} - 1 \\ &< 2\lfloor\sqrt{s}\rfloor + 1. \end{aligned}$$

Since $|P_{r+1}|$ is an odd integer (all augmenting paths have odd length), we can conclude that $|P_{r+1}| \leq 2\lfloor\sqrt{s}\rfloor - 1$. Hence each of P_1, P_2, …, P_r must have length no greater than $2\lfloor\sqrt{s}\rfloor - 1$. Therefore, these r lengths must be distributed among at most $\lfloor\sqrt{s}\rfloor$ different values and this bound can be reached only if $|P_r| = |P_{r+1}|$. Since $|P_{r+1}|$, $|P_{r+2}|$, …, $|P_s|$ cannot contribute more than $s - r = \lceil\sqrt{s}\rceil$ distinct values, the total number of distinct integers in the sequence does not exceed $2\lfloor\sqrt{s}\rfloor$. □

Thus our improved algorithm iterates $\Theta(\sqrt{|V|})$ times, as opposed to $\Theta(|V|)$ times for the original version, a substantial improvement since the worst-case cost of an iteration remains unchanged. For bipartite graphs, we can construct a maximum matching in $O(\sqrt{|V|}\cdot|E|)$ time.

1.2 General matching: blossoms

As mentioned earlier, all of the theorems we proved about matchings hold equally for bipartite and general graphs. Thus, in particular, we can find maximum matchings in general graphs using the same fast algorithm, namely

- Begin with an arbitrary (possibly empty) matching.

- Repeatedly find a maximal set of vertex-disjoint shortest augmenting paths, and use them all to augment the current matching, until no augmenting path can be found.

What does change, however, is that, in a general graph, BFS may fail to find an augmenting path even when one does exist. This failure is due to odd cycles with alternating matched and unmatched edges, where the augmenting path enters the cycle at the node shared by the two adjacent unmatched edges of the cycle (an alternating odd cycle cannot be purely alternating: it must have one pair of consecutive unmatched edges). The augmenting path leaves the cycle at a matched vertex after crossing a matched edge in the cycle to enter an unmatched edge not in the cycle, but the BFS can get to the same vertex by taking the other path around the cycle; if that other path is shorter, then BFS gets to the "exit" vertex by crossing an unmatched edge of the cycle, and, since that vertex is matched, must follow the matched edge, taking it farther around the cycle and missing the exit that is an essential part of the augmenting path. BFS also explores the other path, but because that path is longer, it will encounter vertices already visited by the BFS before it can reach the exit vertex. Thus BFS will report failure on both paths around the cycle and not discover the augmenting path.

Note that the problem is context-dependent: an alternating odd cycle does not necessarily create a problem, as it also requires an augmenting path entering and leaving at the proper places. Jack Edmonds, who pioneered matching algorithms, named these odd cycles *blossoms*—the augmenting path entering the cycle is the stem of the blossom, the vertex common to the two unmatched edges in the cycle is the base of the blossom. Edmonds also showed how to deal with blossoms: simply shrink them, replacing the entire blossom by a single vertex. However, what seems like a simple remedy is anything but: blossoms can be nested, so the vertex representing a blossom shrunk at some earlier step may now be part of a blossom that must be shrunk, and so on. When an augmenting path is finally found in the graph with shrunk blossoms, the blossoms on the path must be re-expanded and the proper path along the blossom selected. Doing all of this requires extra data structures to detect, shrink, maintain, and re-expand blossoms; this introduces a lot of complexity in the algorithm and overhead in the running time. We can still find a maximum matching in $O(\sqrt{|V|} \cdot |E|)$ time, thanks to an algorithm first published by Micali and Vazirani in 1980, but this algorithm is tricky and complicated—to the point where it is one of just 3–4 algorithms to be republished later with additional explanations by another set of authors, because the original paper was so difficult to read!

1.3 Stable marriage

Bipartite matching (and general matching) can also be weighted—every edge has on it a cost function and we can ask for a maximum matching of lowest cost. The approach remains that of augmenting paths, except that now we want augmenting paths of lowest cost, which we can find using a shortest-path greedy algorithm.

Perhaps more interesting is a different optimization version of maximum matching. In this version, we are looking for a *perfect* matching, that is, a matching in which every vertex is matched, and the criterion is stability. Imagine you run a match-making service

and introduce 20 young men and 20 young women with the goal of eventually forming 20 happy couples. After some period of getting acquainted, you eventually ask each woman to rank order all 20 men, from first choice to last choice as a potential husband, and similarly you ask each man to rank order all 20 women, from first choice to last choice as a potential wife. A marriage of woman y to man x is said to be stable if there does not exist any other couple (x',y') such that either woman y prefers x' to her own husband and man x' prefers y to his own wife, or (the symmetric situation) men x prefers y' to his own wife and woman y' prefers x to her own husband. (If either of these two situations occurs, then obviously there an incentive for a swap of spouses, which would remedy the situation—hence the marriages (x,y) and (x',y') would not be stable.) Phrased as a marital endeavor, the problem seems silly, but in fact it has quite a range of applications, although in most applications the lists of preferences will not include all members of the other subset. Perhaps the best know application in the US is "The Match" for medical residencies: medical students about to graduate visit several residency programs (teaching hospitals with openings in this or that area of medicine) all over the USA, while these programs receive visits from a (much larger) number of medical students and interview these students. (Most residency programs have more than one position in each medical area, hence the larger number of students.) At the end of this period of "courtship," each medical student files with a central authority (`www.nrmp.org`) a list of her/his preferences (i.e., a rank ordering of the programs visited), while each residency program files with the same central authority a rank-ordered list of all students who visited. Then. on a fatal day known simply as "MATCH DAY." a computer program computes a stable matching, that is a list of pairs (student, residency position), and both students and residency programs are notified of the outcome. Nearly 40'000 students participate in The Match each year and almost all residency positions are filled through it. The scientists who formulated the stable marriage problem and the scientist who applied it to the medical residency problem received the Nobel prize in economics this year (2012) for their work on this problem. In the next lecture we will study an algorithm for problem of stable marriages and its social implications.