This week we will study *maximum cardinality matchings* in arbitrary undirected graphs. Recall that, given an undirected graph $G = (V, E)$, a matching $M \subseteq E$ is a subset of vertex-disjoint edges. This means the graph (V, M) only contains vertices with degree at most 1. Throughout this handout we we will assume that $|V| = n$ and $|E| = m$.

In *Algorithm Design and Analysis*, we examined the maximum matching problem in bipartite graphs. We viewed it as a network flow problem, which led to a $O(nm)$-time algorithm. There are more efficient algorithms, including the $O(\sqrt{n}m)$ time Hopcroft-Karp algorithm. Neither of these algorithms work for arbitrary graphs.

1 Some History

Jack Edmonds wrote an influential paper called *Paths, Trees, and Flowers* in 1965 that solves the maximum matching problem for arbitrary graphs. To provide some perspective, until Edmonds, nobody wrote very much about the complexity of a problem—if a combinatorial problem was shown to have finitely many solutions then an algorithms existed and the problem was decidable. Edmonds was one of the key players in defining and identifying tractable problems—problems with polynomial-time solutions—and making the case that one should really care if a problem is tractable rather than just finite. In fact, legend has it that when Edmonds found his polynomial time solution to the matching problem, he shouted "Eureka, you shrink!" Edmonds' solution is called the *blossom shrinking* algorithm. This week you will delve into the proof that the algorithm is correct, implement it, and then run a simple test.

Question 1. *Can you think of any natural applications of maximum matchings in arbitrary graphs? Be as specific as possible.*

Question 2. *Why do you think it took until 1965 to truly consider efficiency in solving a problem?*

2 Introduction

Let $G = (V, E)$ be an undirected graph and M be some matching in G. We say a vertex $u \in V$ is *unmatched* in M if u does not belong to any edge in M. We say that a simple path P in G is an *augmenting path* with respect to M if it starts at an unmatched vertex, ends at an unmatched vertex, and its edges belong alternatively to $E \setminus M$ and M. (This definition of an augmenting path mimics the idea of an augmenting path in a flow network). We treat a path as a sequence of edges, rather than as a sequence of vertices.

Given two sets A and B, the *symmetric difference* $A \oplus B$ is defined as $(A \setminus B) \cup (B \setminus A)$, that is, the elements that are in exactly one of the two sets. If M is a matching and P is an augmenting path with respect to M, then the symmetric difference $M \oplus P$ is a matching and $|M \oplus P| = |M| + 1$.

Lemma 1. *Given two matchings M and M^* in G, every vertex in the graph $G' = (V, M \oplus M^*)$ has degree at most 2 so G' is a disjoint union of single vertices, simple paths, or simple cycles. Furthermore, the edges in each simple path or cycle belong alternately to M or M^*.*

Proof. Since M and M^* are matchings, by definition the graphs (V, M) and (V, M^*) contain nodes with degree at most 1. Thus $G' = (V, M \oplus M^*)$ can only increase the degree of each node by at most 1 and therefore each node has degree at most 2. Since each node in G' has degree at most 2, you can enter and leave a node at most once unless it is on a cycle. Thus all edges in G' belong to simple paths or cycles. We claim that if an edge is on a cycle, it must be a simple cycle. For if it were not, then there exists some vertex that joins two cycles necessarily having degree 4, a contradiction. Now, suppose two consecutive edges on a simple path or cycle belong to M (respectively M^*). Then M (respectively M^*) is not a matching, a contradiction. □

Problem 1. *Show that M is a maximum matching with respect to G if and only if there are no augmenting paths with respect to M in G.*

Problem 1 suggests a natural high-level algorithm (See Algorithm 1) for finding a maximum matching in G. Of course, the key is step 2, which actually finds an augmenting path in G. This is the focus of the following two sections.

Algorithm 1 MAXIMUM-MATCHING(G)

Require: An undirected graph G

```
1: $M \leftarrow \emptyset$
2: while there exists an augmenting path $P$ with respect to $M$ do
3:     $M \leftarrow M \oplus P$
4: end while
5: return $M$
```

3 Shrinking Blossoms

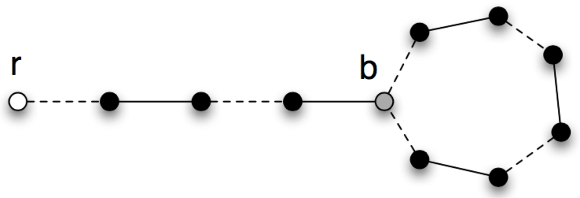

Figure 1: A flower with root r (white node) and blossom base b (grey node) with respect to some matching M on an undirected graph G. The convention is that solid edges are in M and dotted edges are not in M.

A *flower* with respect to a matching M on a graph G is a *stem* together with a *blossom*. A stem is simple path in G that starts at an unmatched vertex r, ends at a matched vertex b, and has edges which alternate consecutively in and out of M. A blossom B is a simple, alternating cycle that starts and ends at b and has first and last edges that are not in M. Figure 1 shows an example flower where dotted edges are not in M and solid edges are in M. The root r is the solid white node and the blossom base b is the grey node.

3.1 Edge Contraction

Given a graph G we can *contract* an edge $(u, v) \in E$ by replacing u and v with a single node uv such that any edge incident to u or v (except the edge (u, v)) is now incident to uv. We eliminate any duplicate edges that may result so G remains a simple graph. Figure 2 provides several example edge contractions.

Problem 2. *Show that edge contraction is commutative. That is, if (u, v) are (x, y) are edges of G, then the graph resulting from the contraction of 1. (u, v) and 2. (x, y) is exactly the same as the graph resulting from the contraction of 1. (x, y) and 2. (u, v). Argue that we can now talk about the contraction of a set of edges $E' \subset E$ on a graph G without worrying about the order of the contraction.*

Problem 3. *Let $G = (V, E)$ be an undirected graph and M be a matching on G. Let B be a blossom in G with blossom base b. Let G_B be the graph obtained by contracting B in G. Similarly, let $M_B = M \setminus B$ be the matching induced by M in G_B. By convention we also let b denote the node in G_B representing the contraction of B in G. Show that G_B contains an augmenting path with respect to M_B if and only if G contains an augmenting path with respect to M.*

4 Finding Augmenting Paths

Here is an outline of Edmonds' blossom shrinking algorithm for finding augmenting paths in G. This procedure is very similar to the solution we gave in *Algorithm Design and Analysis* for finding augmenting paths in bipartite graphs. Let G be an undirected graph and M be a matching on G. Let R be the set of unmatched vertices with

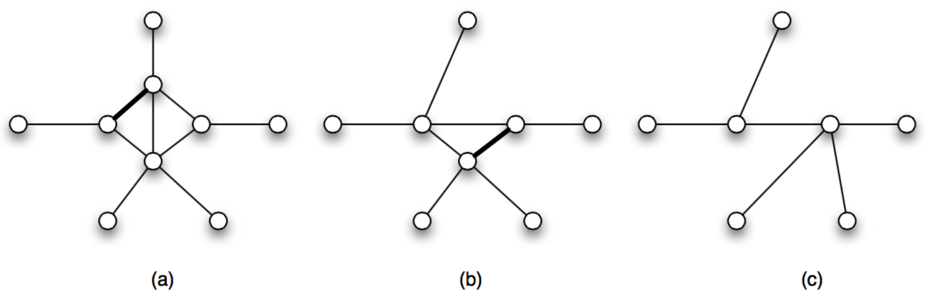

Figure 2: Contracting an edge (u, v) in a graph replaces u and v with a single node w such that any edge incident to u or v (except (u, v)) is now incident to w. To keep the graph simple, we remove duplicated edges. Contracting the dark edge in (a) yields (b). Contracting the dark edge in (b) yields (c).

respect to M. The idea is to perform a complete breadth-first search in G for every $r \in R$ until we either (a) find an augmenting path, or (b) run out of unmatched vertices (in which case we know no augmenting path exists). Given an $r \in R$ we build a BFS tree T, level-by-level, such that edges going from even levels to odd levels are unmatched and edges going from odd levels to even levels are matched.

Begin by placing one of the unmatched nodes r into L_0. Iteratively repeat the following two stages of a (somewhat restricted) breadth-first search. These stages operate on successive levels $L_0, L_1, L_2, \ldots$ where the first stage applies to even numbered levels and the second stage applies to odd numbered levels.

1. EVEN: If node i is in the current level, (i, j) is an unmatched edge in M, and j has not been placed in a previous level, then place j in the next ODD level. Furthermore, If j is unmatched in M, then we have found an augmenting path—i.e. the path from r to j in the BFS tree T is an augmenting path. However, If i is in the current level, (i, j) is an unmatched edge in M, and j was previously placed in an EVEN level then we have found an odd-length cycle which is a blossom B. Problem 3 suggests a good way to deal with this blossom: Contract the blossom B into G_B and completely start the whole augmentation search over again, using G_B instead of G.

2. ODD: If j is in the current level, (i, j) is a matched edge in M, and i has not been placed in a previous level, then place i in the next EVEN level.

At the end of this breadth-first search, if we have not found an augmenting path, then we start again with a new root $r \in R$. If all the root nodes have been exhausted, we know no augmenting path exists, so M must be a maximum matching.

Problem 4. *Argue that there are at most n blossoms (with respect to M) in G.*

Problem 5. *Let G be an undirected graph, M a matching on G, and R a set of unmatched vertices with respect to M. Argue that the sequence of breadth-first searches on R collectively take at most $O(n + m)$ time. That is, after $O(m + n)$ time we have either found an augmenting path, found a blossom, or emptied R.*

Problem 6. *Using your results from the previous two problems, describe an $O((n + m)n^2)$ implementation of Edmonds' blossom shrink algorithm.*

5 Theory v. Practice

The algorithm from Problem 6 is $O(n^4)$-time on dense graphs. This is not great. It can be improved to $O(n^3)$ by not restarting the BFS every time one encounters a blossom. Thus, finding an augmenting path takes $O(n + m)$ time instead of $O((n + m)n)$ time in the worst case. Since $|M| \leq n/2$ we have an $O((n + m)n)$ algorithm. Micali and Vazirani have an $O(\sqrt{n}m)$-time algorithm for finding maximum cardinality matchings in arbitrary graphs. This matches the upper bound given by Karp and Hopcraft. The algorithm is complicated. Slightly better non-combinatorial algorithms based on matrix multiplication exist—these run in time $O(n^\omega)$ where ω [1] is the matrix multiplication constant – however they are not so practical.

Problem 7. *Implement the $O((n+m)n^2)$-time algorithm from Problem 6. Consider implementing a* contract *method for whatever graph class you use. Then your matching algorithm is essentially a loop built around a slightly-specialized BFS. Run your program on the complete graph of 10 vertices and use* `dot` *or* `dotty` *(or any other visualization tool) to illustrate the result.* `dot` *and* `dotty` *are programs which take descriptions of graphs and output computer-drawn images of the graphs. These tools can be incredibly useful when trying to visualize data so it's good to have them in your arsenal. I suggest adding a flag to your program so that it output* `dot` *or* `dotty` *input directly. Use the convention that a matched edge is solid and an unmatched edge is dotted.*

[1]Currently, the best is $\omega = 2.3728639$, due to François le Gall (2014). Chris Umans (Williams '96 CS major) is part of a group of researchers working to show that $\omega = 2$.