Lecture notes on:

Maximum matching in bipartite and non-bipartite graphs

(Draft)

Lecturer: *Uri Zwick* *

June 5, 2013

1 The maximum matching problem

Let $G = (V, E)$ be an undirected graph. A set $M \subseteq E$ is a *matching* if no two edges in M have a common vertex. A vertex v is *matched* by M if it is contained is an edge of M, and *unmatched* otherwise. In the *maximum matching problem* we are asked to find a matching M of maximum size in a given input graph $G = (V, E)$.

The maximum matching problem in *bipartite* graphs can be easily reduced to a maximum flow problem in *unit* graphs that can be solved in $O(m\sqrt{n})$ time using Dinic's algorithm. We present the original derivation of this result, due to Hopcroft and Karp [HK73].

The maximum matching problem in general, not necessarily bipartite, graphs is more challenging. We present here a classical algorithm of Edmonds [Edm65] for solving the problem and discuss its efficient implementation.

2 Alternating and augmenting paths

Definition 2.1 (Alternating paths and cycles) *Let $G = (V, E)$ be a graph and let M be a matching in M. A path P is said to be an alternating path with respect to M if and only if among every two consecutive edges along the path, exactly one belongs to M. An alternating cycle C is defined similarly.*

*School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E–mail: zwick@tau.ac.il

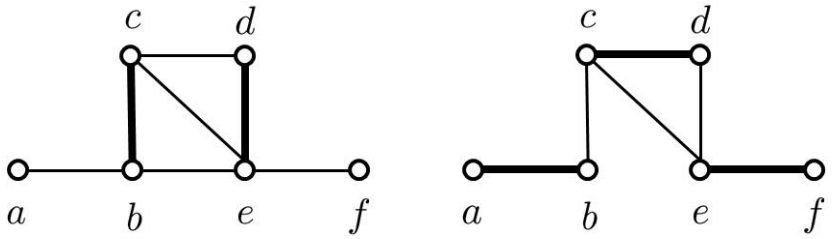

Figure 1: Matchings in a simple graph

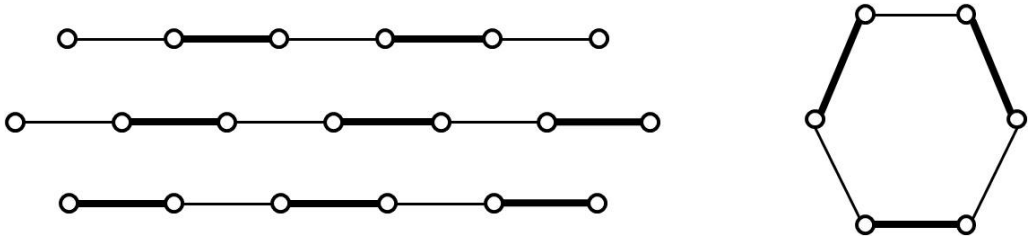

Figure 2: Alternating paths and cycles

Some alternating paths and an alternating cycle are shown in Figure 2. We use the convention that edges that belong to a matching M are shown as *thick* edges, while edges not belonging to M are shown as *thin* edges.

Definition 2.2 (Symmetric difference) *If A and B are sets, we let $A \oplus B = (A - B) \cup (B - A)$ be their* symmetric difference.

The following lemma is now obvious.

Lemma 2.3 *If M is a matching and P is an alternating path with respect to M, where each endpoint of P is either unmatched by M or matched by the edge of P touching it, then $M \oplus P$ is also a matching.*

Note that if P starts and ends in vertices unmatched by M (e.g., the top path in Figure 2), then $|M \oplus P| = |M| + 1$, i.e., $M \oplus P$ is a larger matching. If P starts with an edge that does not belong to M and ends with an edge of M (e.g., the middle path in Figure 2), then $|M \oplus P| = |M|$. Finally, if P starts and ends with edges of M (see the bottom path in Figure 2), then $|M \oplus P| = |M| - 1$.

Definition 2.4 (Augmenting paths) *An* augmenting path *P with respect to a matching M is an alternating path that starts and ends in* unmatched *vertices.*

We have seen above that a matching that admits an augmenting path is not a maximum matching. We show below that the converse also holds. We start with the following simple observation.

Lemma 2.5 *Let $G = (V, E)$ be an undirected graph and let M_1 and M_2 be matchings in G. Then, the subgraph $(V, M_1 \oplus M_2)$ is composed of isolated vertices, alternating paths and alternating cycles with respect to both M_1 and M_2.*

Lemma 2.6 *Let $G = (V, E)$ be an undirected graph and let M and M' be matchings in G such that $|M'| = |M| + k$, where $k \geq 1$. Then, there are at least k vertex disjoint augmenting paths in G with respect to M. At least one of these augmenting paths is of length at most $\frac{n}{k} - 1$.*

Proof: Consider the subgraph $(V, M \oplus M')$. By Lemma 2.5 it consists of alternating paths and cycles with respect to both M and M'. The set $M \oplus M'$ contains exactly k more edges from M' than from M. It must then contain at least k vertex disjoint paths that start and end with edges of M'. These paths are augmenting paths with respect to M. At least one of these paths contains at most $\frac{n}{k}$ vertices and is its length, therefore, is at most $\frac{n}{k} - 1$. □

Theorem 2.7 *Let $G = (V, E)$ be an undirected graph and let M be a matching in G. Then, M is a maximum matching in G if and only if there are no augmenting paths with respect to M in G.*

Proof: If P is an augmenting path with respect to M, then $M \oplus P$ is also a matching and $|M \oplus P| > |M|$, so M is not a maximum cardinality matching of G. If M is not a maximum matching, then by Lemma 2.6 there is at least one augmenting path with respect to M. □

Theorem 2.7 suggests the following simple algorithm for finding a maximum matching. Start with some initial matching M, possibly the empty one. As long as there is an augmenting path P with respect to M, augment M using P and repeat. All that remains, therefore, is to devise a procedure for finding augmenting paths, if they exist.

3 Alternating trees

In this section we try a natural approach for finding an augmenting path with respect to a matching M, if one exists. The approach will work properly in bipartite graphs. We will extend it later so that it would also be able to cope with non-bipartite graphs.

We construct a forest of *alternating trees*. An alternating tree is a tree whose root is an unmatched vertex. All root-leaf paths in an alternating tree are alternating paths with respect to M. If we manage to add an unmatched vertex, other than the root, to an alternating tree, we have identified an augmenting path.

Vertices at the even levels of alternating tree are labeled EVEN while vertices at the odd levels are labeled ODD. Vertices that are currently in no alternating tree are unlabeled.

Initially all nodes are unlabeled and all edges are unexplored. In each step we either choose an unlabeled and unmatched vertex r, label it EVEN, and make it the root of a new tree, or we choose an unexplored edge (u, v) where u is EVEN, and explore it as follows:

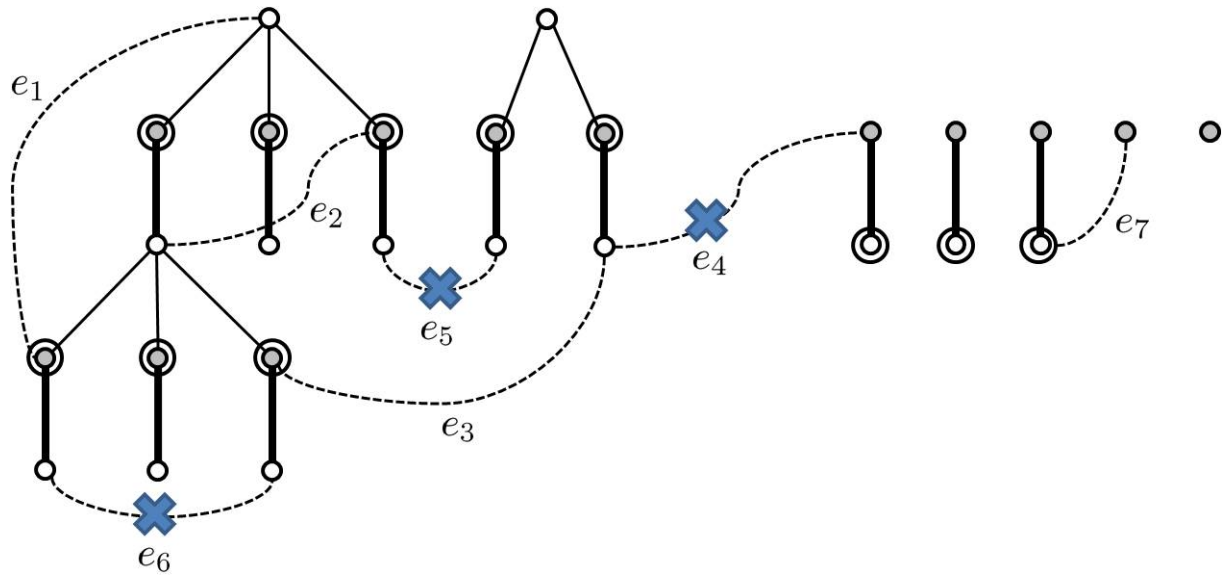

Figure 3: A forest of alternating trees

- If v is unmatched, we have found an augmenting path.
- If v is unlabeled and matched, we let w be the vertex matched to v, i.e., $(v, w) \in M$. (It is easy to check that w must also be unlabeled.) We add v and w, and the edges (u, v) and (v, w), to the tree and label v ODD and w EVEN.
- If v is already labeled ODD, we do nothing. (We simply found an alternative odd length alternating path from an unmatched vertex to v. Note that u and v are not necessarily in the same alternating tree.)
- If v is labeled EVEN and does not belong to the tree containing u, we discovered an augmenting path composed of the path from the root of u's tree to u, the edge (u, v), and then the path from v to the root of its tree.
- If v is labeled EVEN and belongs to the same tree as u, we discovered an odd length cycle in the graph. Such an odd cycle will be called a *blossom*. Note that this case cannot occur in bipartite graphs.

The next to last case, in which u and v are EVEN vertices belonging to different trees, can be avoided by exploring all edges touching the EVEN vertices of a tree before starting to construct the next tree.

The process stops either when an augmenting path or a blossom are found, or when there are no more unlabeled and unmatched vertices or unexplored edges touching at least one EVEN vertex.

Lemma 3.1 *If there is an augmenting path with respect to M, then the above process finds an augmenting path or a blossom. The running time is $O(m)$.*

This immediately gives an $O(mn)$-time algorithm for finding a maximum matching in bipartite graphs. We improve this to $O(m\sqrt{n})$ in Section 5.

4 Minimum vertex cover in bipartite graphs

Definition 4.1 (Vertex Cover) *Let $G = (V, E)$ be an undirected graph. A set $C \subseteq V$ is said to be a* vertex cover *if and only if $e \cap C \neq \phi$, for every $e \in E$.*

In other words, C is a vertex cover if and only if every edge $(u, v) \in E$, either $u \in C$ or $v \in C$ (or both). In the *minimum vertex cover* problem we are given an undirected graph $G = (V, E)$ and asked to find a vertex cover of minimum size. For general graphs, the problem is NP-complete. (Note that C is a vertex cover if and only if $V - C$ is an independent set.) In this section we show, perhaps surprisingly, that the minimum vertex cover problem in bipartite graphs can be efficiently solved. It is solved, in fact, by the algorithm sketched in the previous section. Furthermore, the size of a minimum vertex cover is equal to the size of a maximum matching.

Theorem 4.2 *Let $G = (V, E)$ be a bipartite graph. The size of a maximum matching in G is equal to the size of a minimum vertex cover of G.*

Proof: Let M be a matching in G and let C be a vertex cover of G. It is clear that $|M| \leq |C|$, as any vertex cover must contain at least one endpoint of each edge of the matching M.

Let M is a maximum matching in G. We show that there exists a vertex cover C of G such that $|M| = |C|$.

Suppose that $V = S \cup T$, where $S \cap T = \phi$ and that all edges connect vertices in S with vertices of T. We use the generic algorithm of the previous section to construct a fully grown forest F of alternating trees with respect to M rooted at the unmatched vertices in S. Note that all vertices of S contained in F are EVEN while all vertices of T contained in F are ODD.

Let $M' = M \cap F$ be the edges of M that are contained in F, and let $M'' = M - F$ be the edges of M that are not contained in F. We construct a vertex cover C by taking the endpoints of the edges of M' that belong to T and the endpoints of the edges of M'' that belong to S. Clearly $|C| = |M|$, as we are choosing exactly one vertex from each edge of M. The set C contains all vertices of T that belong to F and all vertices of S that do not belong to F. (See Figure 3.) We claim that C is a vertex cover. Let $(u, v) \in E$, where $u \in S$ and $v \in T$. If $u \notin F$, then $u \in C$. If $u \in F$, then u is an EVEN vertex. As all edges out of EVEN vertices are explored, we get that v must be an ODD vertex and thus $v \in C$. □

The proof of Theorem 4.2 gives us an efficient algorithm for finding a minimum vertex cover of a bipartite graph $G = (V, E)$: Find a maximum matching in G, construct a forest F of alternating trees and then a minimum vertex cover C. Note that the last two steps can be performed in $O(m)$ time.

Theorem 4.2 also follows easily from the max-flow min-cut theorem.

5 The Hopcroft-Karp algorithm for bipartite graphs

Let $G = (S, T, E)$ be a bipartite graph and let M be a matching in G. Instead of building the alternating trees one at a time, we can use breadth-first search to simultaneously build alternating trees from all unmatched vertices of S. This allows us to find *shortest* augmenting paths. Furthermore, we can find a *maximal* collection of vertex disjoint shortest augmenting paths.

The Hopcroft-Karp algorithm [HK73] works in *phases*. In each phase it constructs a maximal collection of vertex disjoint shortest augmenting paths and uses them to augment the matching.

Lemma 5.1 *Let P be a shortest augmenting path with respect to M and let P' be an augmenting path with respect to $M \oplus P$. Then $|P'| \geq |P| + 2|P \cap P'|$.*

Proof: Let $N = (M \oplus P) \oplus P'$. Then, N is a matching and $|N| = |M| + 2$. By Lemma 2.6, $M \oplus N$ contains two vertex disjoint augmenting paths P_1 and P_2 with respect to M. As P is a shortest augmenting path with respect to M we get that $|P_1|, |P_2| \geq |P|$. Since $M \oplus N = P \oplus P'$, we get that $|P \oplus P'| \geq |P_1| + |P_2| \geq 2|P|$. Now $|P \oplus P'| = |P| + |P'| - 2|P \cap P'|$. Hence $|P'| \geq |P| + 2|P \cap P'|$. □

Lemma 5.2 *Let $G = (S, T, E)$ be a bipartite graph and let M be a matching in G. Suppose that $\mathcal{P} = \langle P_1, P_2, \ldots, P_\ell \rangle$ is a maximal collection of disjoint shortest augmenting paths with respect to M. Let $M' = M \oplus P_1 \oplus P_2 \oplus \cdots \oplus P_\ell$ and let P' be an augmenting path with respect to M'. Then, $|P'| > |P_1| = \ldots = |P_\ell|$.*

Proof: If P' does not intersect any path from $\mathcal{P}$, then by the maximality of $\mathcal{P}$ we have $|P'| > |P_1| = \ldots = |P_\ell|$. Otherwise, let P_i be the last path from $\mathcal{P}$ that intersects P'. Then, P_i is an shortest augmenting path with respect to $M_{i-1} = M \oplus P_1 \oplus P_2 \oplus \cdots \oplus P_{i-1}$ and P' is an augmenting path with respect to $M_{i-1} \oplus P_i$. By Lemma 5.1, we get that $|P'| \geq |P_i| + 2|P' \cap P_i| > |P_i|$. □

Theorem 5.3 *The Hopcroft-Karp algorithm finds a maximum matching in a bipartite graph after at most $2\sqrt{n}$ phases.*

Proof: By Lemma 5.2, we get that the shortest augmenting paths found in each phase get longer and longer. Thus, after $\sqrt{n}$ phases, if the algorithm is not done yet, the lengths of the shortest augmenting paths found is at least $\sqrt{n}$. By Lemma 2.6, the fact that the length of the shortest augmenting path is at least $\sqrt{n}$ implies that the size of the current matching is smaller than the size of the maximum matching by at most $\sqrt{n}$. Thus, a maximum matching would be found after at most $\sqrt{n}$ additional iterations. □

In the bipartite case, finding a maximal collection of disjoint shortest augmenting paths is similar to finding a blocking flow in a layered network. It can easily be done in $O(m)$ time. We omit the details.

6 Edmonds' algorithm for non-bipartite graphs

We now return to the problem of finding a maximum matching in non-bipartite graphs.

Definition 6.1 (Flowers and blossoms) *A* flower *with respect to a matching M is composed of a* stem*, which is an alternating path of even length from an unmatched vertex r, called the* root*, to vertex b, and an 'alternating' cycle of odd length that passes through b, called a* blossom*. The last edge on the stem belongs to M. The two edges of the blossom touching b are not in M. Other than that, every second edge on the blossom belongs to M. The vertex b is said to be the* base *of the blossom.*

We stress that an odd alternating cycle is considered to a blossom only if it is the blossom of some flower.

When the algorithm of Section 3 finds an edge $(u, v) \in E$ such that u and v are EVEN vertices of the same tree, it, in fact, finds a flower. Edmonds [Edm65] suggests *shrinking* (i.e., contracting) the blossom B of the flower into a new vertex, labeling this new vertex EVEN, and continuing. We let G_B be the graph obtained by contracting B, and $M_B = M - B$ the matching induced by M in this new graph. We use B to denote both the edge set of the blossom B and the vertex of G_B obtained by contracting B.

Lemma 6.2 *Let $G = (V, E)$ be an undirected graph and let M be a matching in G. Let B be a blossom in G. If there is an augmenting path with respect to M_B in G_B, then there is also an augmenting path with respect to M in G.*

Proof: Let P be an augmenting path in G_B with respect to M_B. If P does not pass through B, then P is also an augmenting path in G with respect to M, and we are done. Suppose, therefore, that P does pass through B. We consider two cases:

Case 1: P starts (or ends) at B.

As the augmenting path P starts at B, we get that the vertex B is unmatched in G_B. It follows that the base b of B is unmatched in G. Let (B, c) be the first edge on P and let P_c be the part of P that starts at c. Clearly $(B, c) \notin M_B$. Also, there is a vertex $v \in B$ such that $(v, c) \in E$ and $(v, c) \notin M$. In the blossom B, we can find an even length alternating path Q from b to v. This path ends with an edge of M. The path $Q, (v, c), P_c$ is then an augmenting path in G with respect to M, as required.

Case 2: B is not the first or last vertex on P.

Let $(a, B), (B, c) \in P$ be the edges of P that touch B. Assume that $(a, B) \in M_B$ and $(B, c) \notin M_B$. Let P_a be the part of P up to a, and let P_c be the part of P from c. As $(a, B) \in M_b$, the edge (a, b) must be present in the original graph and $(a, b) \in M$, as the only edge of the matching that enters a blossom enters it at its base. As before, there is a vertex $v \in B$ such that $(v, c) \in E$ and

$(v, c) \notin M$. We can again find an even alternating path Q in the blossom from b to v. The path $P_a, (a, b), Q, (v, c), P_c$ is then an augmenting path in G with respect to M, as required. □

Lemma 6.3 *Let $G = (V, E)$ be an undirected graph and let M be a matching in G. Let B be a blossom in G which is part of a flower with stem Q. If there is an augmenting path with respect to M in G, then there is also an augmenting path with respect to M_B in G_B.*

Proof: Let b be the base of the blossom B. We consider two cases:

Case 1: b is unmatched by M.

Let P be an augmenting path in G with respect to M. If P does not pass through any vertex of B, then P is also an augmenting path with respect to M_B in G_B. Suppose therefore that P does pass through B. As P starts and ends at unmatched vertices, and only one vertex on B is unmatched, we may assume that P does not start on B. Let P' be the prefix of P ending at the first encounter of P with a vertex of B. Let $P' = P'', (a, v)$, where $v \in B$. Then, $P'', (a, B)$ is an augmenting path with respect to M_B in G_B, as B is unmatched by M_B.

Case 2: b is matched by M.

Let $M' = M \oplus Q$, where Q is a stem of a flower with blossom B. As Q is an even alternating path starting at an unmatched vertex, we get that M' is a matching and that $|M'| = |M|$. Note that B is also a blossom with respect to M' and that b is unmatched with respect to M'. We also have $|M'_B| = |M_B|$.

As M' and M are matching of the same size and as there is an augmenting path with respect to M, there must also be an augmenting path with respect to M'. As b in unmatched by M', we get by Case 1 that there must be augmenting path with respect to M'_B in G_B, and as M'_B and M_B are again of the same size, we get that there must also be an augmenting path with respect to M_B in G_B, as required. □

We can therefore modify the algorithm of Section 3 for finding augmenting paths as follows. We start constructing alternating trees from the unmatched vertices. If a blossom B is discovered, we shrink it. The blossom B is now considered to be an EVEN vertex so we can now explore all edges emanating from the ODD vertices contained in B. At some stage a new blossom B' may be discovered. Note that B can be one of the vertices that participate in B'. The algorithm terminates either when an augmenting path is found, or when all unmatched vertices are made roots of trees and all edges touching EVEN vertices are explored.

Although conceptually simple, the efficient implementation of Edmonds' blossom shrinking algorithm is a non-trivial task. We discuss this issue further in Section 8.

7 Odd vertex covers

Definition 7.1 (Odd Vertex Cover) *Let $G = (V, E)$ be an undirected graph. An* odd vertex cover *of G is composed of a subset $C \subseteq V$ of vertices and a collection of subsets $B_1, B_2, \ldots, B_k \subseteq V$ such that for every edge $(u, v) \in E$ either $u \in C$ or $v \in C$, or there is an index $1 \leq i \leq k$ for which $u, v \in B_i$. The* size *of such a cover is defined to be $|C| + \sum_{i=1}^{k} \lfloor \frac{|B_i|}{2} \rfloor$. We may assume that the sets $B_1, B_2, \ldots, B_k$ are of odd size.*

Theorem 7.2 *Let $G = (V, E)$ be an undirected graph. The size of a maximum matching in G is equal to the size of a minimum odd vertex cover of G.*

8 Efficient implementation of Edmonds' algorithm

We represent the current matching M using an attribute *mate* defined for all vertices of the graph. If $(v, w) \in M$, then $mate[v] = w$ and $mate[w] = v$. If v is unmatched by M, we let $mate[v] = null$.

`alternatingforest`($mate$) constructs a tree of alternating trees with respect to M, shrinking blossoms whenever they are encountered, until either an augmenting path is found, or no more vertices and edges can be added to the forest

Each vertex $v \in V$ has two additional attributes $pred[v]$ and $label[v]$. If v is contained in an alternating tree, then $pred[v]$ is its parent, and $label[v]$ says whether it is an EVEN or ODD vertex. If v is not contained in any tree, then $pred[v]$ and $label[v]$ are both $null$.

Blossoms encountered while building the alternating trees are not explicitly contracted. Instead, we use a *union-find* data structure to maintain for each vertex v the base of the blossom to which it currently belongs.

A union-find data structure maintains a collection of disjoint sets. It supports the following operations:

`make-set`(v) – Create a singleton set $\{v\}$ containing the element v.

`union`(v, w) – Unite the sets currently containing the elements v and w.

`find`(v) – Returns a representative element of the set currently containing v.

`select`(v) – Make v the representative element of its set.

An edge $(v, w) \in E$ in the original graph corresponds to an edge (`find`(v), `find`(w)) in the implicitly contracted graph. (Note that `find`(v) is either v, if v was not swallowed by a blossom yet, or the base of its blossom if it had.)

`alternatingforest`($mate$) starts by letting $label[v] = null$ and $pred[v] = null$, and by building a singleton set $\{v\}$ using `make-set`(v), for every $v \in V$. It then goes over the vertices of the graph one by one. If v is still unmatched by the time it is examined, an alternating tree is grown from it using `alternatingtree`(v).

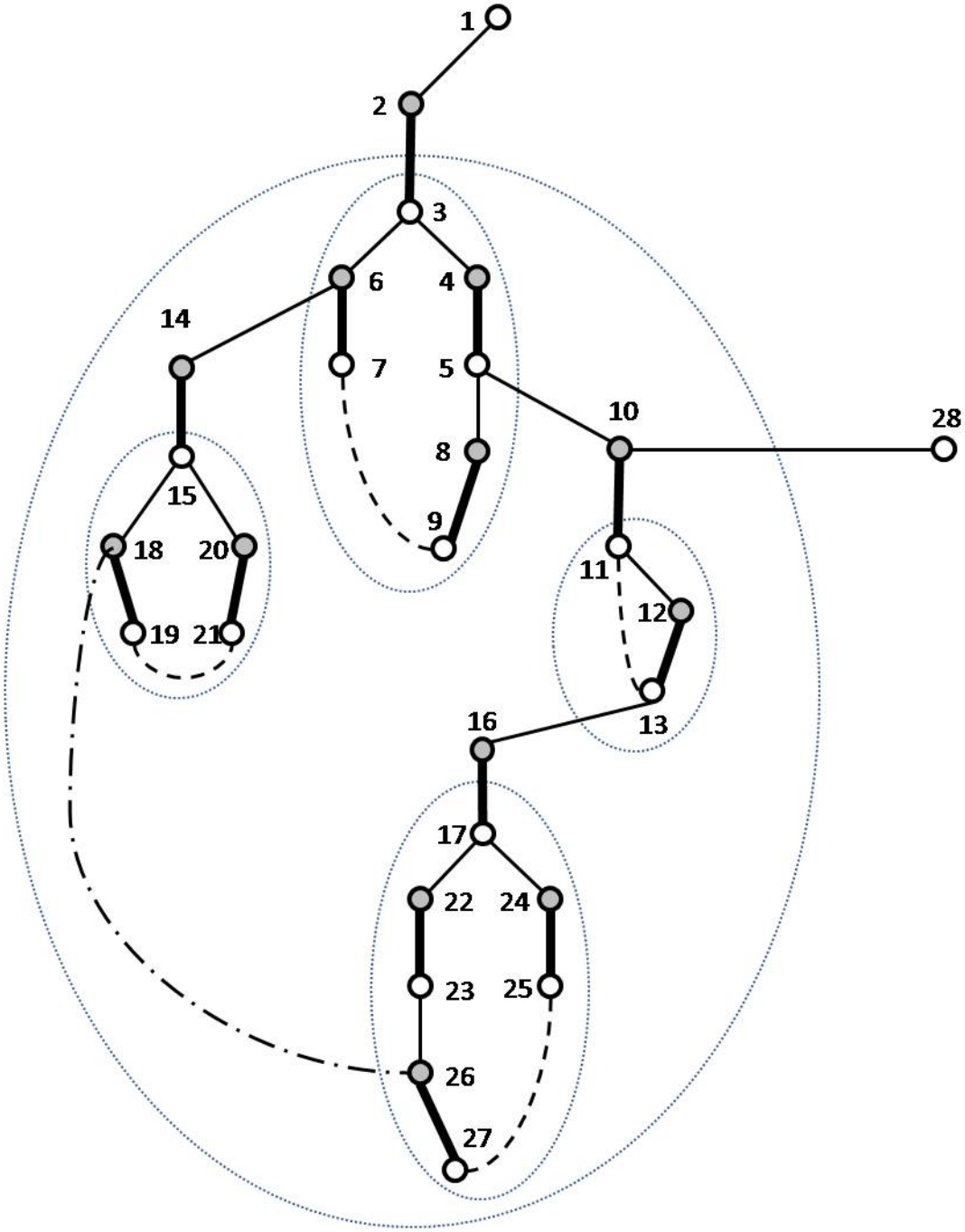

Figure 4: Blossoms shrinking example.

Function alternatingforest($mate$)

foreach $v \in V$ **do**
  $label[v] \leftarrow null$
  $pred[v] \leftarrow null$
  `make-set`(v)
foreach $v \in V$ **do**
  if $mate[v] = null$ **then**
    `alternating-tree`(v)

Function alternatingtree(v)

$label[v] \leftarrow$ EVEN
$Q \leftarrow \langle v \rangle$
while $Q \neq \emptyset$ **do**
  $v \leftarrow$ `extract`(Q)
  foreach $(v, w) \in E$ **do**
    `examine`(v, w)

Function examine(v, w)

$\bar{v} \leftarrow$ `find`(v)
$\bar{w} \leftarrow$ `find`(w)
if $\bar{v} \neq \bar{w}$ **then**
  if $label[\bar{w}] = null$ **then**
    if $mate[w] = null$ **then**
      `augmenting-path`(v, w)
    else
      `extend-tree`(v, w)
  else if $label[\bar{w}] =$ EVEN **then**
    `shrink-blossom`(v, w)

Function extendtree(v, w)

$label[w] \leftarrow$ ODD
$pred[w] \leftarrow v$
$label[mate[w]] \leftarrow$ EVEN
$pred[mate[w]] \leftarrow w$
`insert`$(Q, mate[w])$

Function shrinkblossom(v, w)

$b \leftarrow$ `find-base`(v, w)
`shrink-path`(b, v, w)
`shrink-path`(b, w, v)

Function shrinkpath(b, v, w)

$u \leftarrow$ `find`(v)
while $u \neq b$ **do**
  `union`(b, u)
  $u \leftarrow mate[u]$
  `union`(b, u)
  `select`(b)
  `insert`(Q, u)
  $bridge[u] \leftarrow (v, w)$
  $u \leftarrow$ `find`$(pred[u])$

Function augmentingpath(v, w)

$r \leftarrow$ `find-root`(v)
throw $\langle w \rangle +$ `find-path`(v, r)

Function findpath(s, t)

if $s = t$ **then**
  return $\langle s \rangle$
else if $label[s] =$ EVEN **then**
  return $\langle s, mate[s] \rangle +$ `find-path`$(pred[mate[s]], t)$
else
  $(v, w) \leftarrow bridge[s]$
  return $\langle s \rangle +$ `reverse`(`find-path`$(v, mate[s])) +$ `find-path`(w, t)

`alternatingtree`(v) uses a queue Q to store all EVEN vertices, or ODD vertices contained in blossoms, that were not explored yet. Initially $Q = \langle v \rangle$. As long as Q is not empty, a vertex v is removed from Q and each edge $(v, w) \in E$ incident on it is examined by `examine`(v, w).

`examine`(v, w) first lets $\bar{v} \leftarrow$ `find`(v) and $\bar{w} \leftarrow$ `find`(w) be the bases of the blossoms currently containing v and w. If $\bar{v} = \bar{w}$, then the edge is a self-loop in the contracted graph and is therefore ignored. Assume, therefore, that $\bar{v} \neq \bar{w}$. If $label[\bar{w}] = null$, then w was not added to the forest yet. (In particular, we have $\bar{w} = w$.) If $mate[w] = null$, then w is unmatched and we have found an augmenting path. (That was easy!) Reconstructing this augmenting path is a non-trivial task handled by `augmentingpath`(v, w). If w is not yet in the forest (i.e., $label[w] = null$), but it is matched (i.e., $mate[w] \neq null$), we add w and $mate[w]$ to the current tree using `extendtree`(v, w). If $label[\bar{w}]$ is EVEN, we discovered a new blossom. We (implicitly) shrink this blossom using `shrinkblossom`(v, w). If $label[\bar{w}]$ is ODD, nothing needs to be done.

`extendtree`(v, w) is straightforward. We add w and $mate[w]$ to the tree and label them ODD and EVEN, respectively. We add $mate[w]$ to Q, the queue containing the vertices that still need to be explored.

`findpath`(s, t) finds a simple even alternating path starting at s and ending at t, assuming that the simple even alternating path from s to the root of its tree passes through t after an even number of steps.

`shrinkblossom`(v, w) shrinks the blossom formed by the edge (v, w). It starts by computing the base b of the blossom using a call $b \leftarrow$ `find-base`(v, w). `find-base`(v, w) climbs up the tree until finding the least common ancestor of v and w. (The code of `find-base`(v, w) is omitted.) The paths from v to b and from w to b are then shrunk using the calls `shrinkpath`(b, v, w) and `shrinkpath`(b, w, v). The edge (v, w) (and its reverse (w, v)) is said to be the *bridge* of the blossom. `shrinkpath`(b, v, w)

visits all bases on the path from v to b and unites them with b. (This is done using `union`(b, u). The call `select`(b) makes sure that b is the representative element of the newly formed blossom.) For every ODD vertex u on the path from v to b, we set $bridge[u] \leftarrow (v, w)$.

As an example, consider the evaluation of an even alternating path from vertex 10 to vertex 1, the root of its tree, using a call `findpath`$(10, 1)$ in the example of Figure 4. As 10 is ODD and $bridge[10] = (26, 18)$, the path returned by the call is:

$$\langle 10\rangle + \texttt{reverse}(\texttt{findpath}(26, 11)) + \texttt{findpath}(18, 1)$$

Consider now `findpath`$(26, 11)$. As 26 is again ODD, and $bridge[26] = (27, 25)$, we get $\langle 26\rangle +$ `reverse`(`findpath`$(27, 27)) +$ `findpath`$(25, 11)$. As 25 is EVEN, `findpath`$(25, 11)$ expands to $\langle 25, 24, 17, 16, 13, 12, 11\rangle$. Thus, `findpath`$(26, 11)$ is the path $\langle 26, 27, 25, 24, 17, 16, 13, 12, 11\rangle$. In a similar manner, it can be seen that `findpath`$(18, 1)$ returns $\langle 18, 19, 21, 20, 15, 14, 6, 7, 9, 8, 5, 4, 3, 2, 1\rangle$. Putting everything together, we get that the path returned by `findpath`$(10, 1)$ is

$$\langle 10, 11, 12, 13, 16, 17, 24, 25, 27, 26, 18, 19, 21, 20, 15, 14, 6, 7, 9, 8, 5, 4, 3, 2, 1\rangle.$$

The maximum matching problem is general graph can actually be solved in $O(m\sqrt{n})$ time (see [MV80], [GT91], [Vaz94]), matching the time bound for bipartite matching, but this is beyond the scope of these notes.

Exercises

Exercise 1

(a) Let $G = (S, T, E)$ be a bipartite graph. Let M_1 and M_2 be two matchings in G. Show that there is always a matching that matches all the vertices of S matched by M_1 *and* all vertices of T matched by M_2.

(b) Let $G = (V, E)$ be a graph and let M be a matching in G. Show that there is always a *maximum* matching M' that matches all the vertices matched by M.

Exercise 2 Let $G = (V, E)$ be a graph and let M be a matching in G. Suppose that v is unmatched by M and that there is no augmenting path with respect to M that start at v. Show that there exists a maximum matching M^* in which v is unmatched.

Exercise 3 Let $G = (S, T, E)$ be a bipartite graph and let k be the size of the maximum matching in G. Show that the algorithm of Hopcroft and Karp actually finds a maximum matching in G in $O(m\sqrt{k})$ time, where $m = |E|$.

Exercise 3 Let $G = (V, E)$ be a non-bipartite graph and B a *flowerless blossom* with respect to a matching M, i.e., a blossom that does not belong to any flower. Is it true that there is an

augmenting path with respect to M in G if and only if there is an augmenting path with respect to M_B in G_B?

Exercise 4 The purpose of this exercise is to give an alternative, perhaps more "direct", proof of Lemma . Let $G = (V, E)$ be a graph. Let M be a matching in G. Let B be a blossom with respect to M with stem $Q = \langle u_0, u_1, \ldots, u_{2k} \rangle$. Let P be an augmenting path with respect to M in G. Suppose that P does not start at u_0. (If it does, we consider $\bar{P}$, the path obtained by reversing P. Our goal is to show that there exists an augmenting path with respect to M_B in G_B. Let v be the first vertex on P that also lies in $Q \cup B$. (If P and $Q \cup B$ are disjoint, then P is also an augmenting path with respect to M_B in G_B and we are done.)

(a) Show that if $v \in B$, then an augmenting path with respect to M_B in G_B can be easily found.

(b) Show that if $v = u_{2i}$, for some $1 \leq i \leq k$, then an augmenting path with respect to M_B in G_B can be easily found.

(c) Suppose that $v = u_{2i-1}$, for some $1 \leq i \leq k$. Let $Q' = P_v Q^v$, where P_v is the prefix of P that reaches v, and Q^v is suffix of Q that starts at v. Show that Q' can also serve as a stem for B, and that $\bar{P}$ and Q' do not start at the same vertex.

(d) Show that by repeating the operation described in (c), i.e., replacing P and Q by $\bar{P}$ and $P_v Q^v$, we must eventually find a stem Q' and an augmenting path P', which is either P or $\bar{P}$, for which either (a) or (b) hold. (Hint: show that $|Q^v|$ gets smaller in each iteration, i.e., the augmenting path hits the stem closer and closer to the blossom.)

Exercise 5 Let $G = (V, E)$ be a directed graph with a *length* function $\ell : E \to \mathbb{R}$ defined on its edges. Construct an undirected bipartite graph $G' = (V_1 \cup V_2, E')$ where $V_1 = \{v_1 \mid v \in V\}$ and $V_2 = \{v_2 \mid v \in V\}$ and $E' = \{(u_1, v_2) \mid (u, v) \in E\} \cup \{(u_1, u_2) \mid u \in V\}$, and define a weight function $w : E' \to \mathbb{R}$ as follows: $w(u_1, v_2) = \ell(u, v)$, for every $(u, v) \in E$, and $w(u_1, u_2) = 0$, for every $u \in V$. Show that G has a negative cycle if and only if G' has a perfect matching of negative weight.

References

[Edm65] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[GT91] H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM*, 38(4):815–853, 1991.

[HK73] J.E. Hopcroft and R.M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[MV80] S. Micali and V.V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Proc. of 21st FOCS*, pages 17–27, 1980.

[Vaz94] V.V. Vazirani. A theory of alternating paths and blossoms for proving correctness of the $O(\sqrt{V}E)$ general graph maximum matching algorithm. *Combinatorica*, 14(1):71–109, 1994.