CS577: Homework 8

Haruki Yamaguchi
hy@cs.wisc.edu

Keith Funkhouser
wfunkhouser@cs.wisc.edu

November 12th, 2015

1 Algorithm description

1.1 Pseudocode

Input: Matchings M_L and M_R along with the vertex sets they cover, $L' \subseteq L$ and $R' \subseteq R$ (where the original graph is $G = (L \cup R, E)$).
Output: A matching M covering $L' \cup R'$.

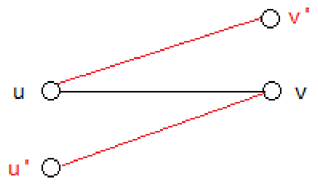
```
  /* Initialize to edges in both $M_L$ and $M_R$ */
1 $M \leftarrow M_L \cap M_R$
  /* Remove edges which are in common from $M_L$ and $M_R$ */
2 $M_L \leftarrow M_L \setminus (M_L \cap M_R)$
3 $M_R \leftarrow M_R \setminus (M_L \cap M_R)$
  /* Add to solution edges which share no common vertices in $M_L$ and $M_R$ */
4 $M \leftarrow M \cup (M_L \oplus M_R)$
  /* Remove edges which share no common vertices from $M_L$ and $M_R$ */
5 $M_L \leftarrow M_L \setminus (M_L \oplus M_R)$
6 $M_R \leftarrow M_R \setminus (M_L \oplus M_R)$
  /* The only remaining edges in $M_L$, $M_R$ are those which share common vertices */
7 Pick edges in $M_L$, $M_R$ which complete the matching
```

1.2 Explanation

(All further edge notation of the form (u, v) can be assumed to have $u \in L$, $v \in R$).

We begin by noting that there exist edges in M_L and M_R which are easier to deal with than others. Consider those edges (u, v) which are identical in both M_L and M_R. These edges can be immediately added to M, since in the final matching we must cover vertices u and v, and there are no other edges in M_L or M_R which are incident upon them (Figure 1). Proof: suppose there were some other edge (u, v') in M_L or M_R. Since (u, v) is in both M_L and M_R, neither can also have (u, v') or (u', v) by definition of a matching to have no more than one edge incident on the same vertex.

Figure 1: For each edge $(u, v) \in M_L \cap M_R$, there cannot exist other edges (u, v') or (u', v) in M_L or M_R, as they would violate the premise that M_L and M_R are matchings.



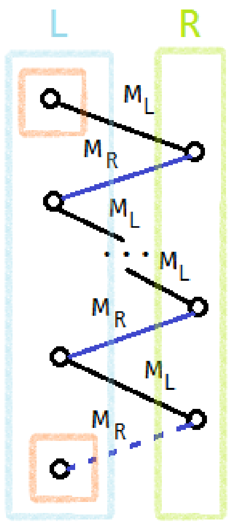
After determining which edges are in common between M_L and M_R, we update M to contain these edges (line 1). Then, we update M_L and M_R so that they no longer contain these edges (line 2).

Another set of edges which are easy to deal with are remaining vertices which contain no common vertices with other edges (note that edges within M_L and M_R, respectively, by definition share no common edges with each other, so these are edges in M_L which share no common vertices with edges in M_R, or vice versa). Equivalently, we can say that these are edges $(u, v) \in (L' \times (R \setminus R')) \cup ((L \setminus L') \times R')$ (proof: suppose (u, v) were an edge with $u \in L'$ and $v \in R'$. Then, the only way that (u, v) can share no common vertices with other edges in M_L and M_R is if $(u, v) \in M_L$ and $(u, v) \in M_R$, but we have already removed vertices satisfying this condition in line 2).

Since these edges share no common vertices and have one vertex in L' or R', we can add them to the matching M (line 4; note: we are using the symbol $\oplus$ here to denote the "no common vertices" operation) by the same logic with which we added edges in $M_L \cap M_R$, and update M_L and M_R accordingly (line 5).

Consider the only remaining edges in $M_L \cup M_R$. These edges are exactly those which share at least one vertex in common with another edge. The "shape" of these remaining vertices is very specific, however, in that the vertices can be broken down into connected components of size $E \geq 3$, of the shape shown in Figure 2:

Figure 2: The general shape of each of the connected components remaining in $M_L \cup M_R$. At most one of the vertices in orange can be in L'.



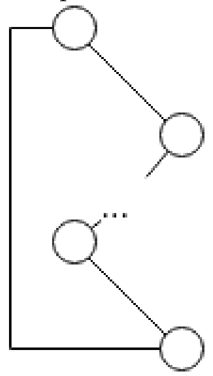
Suppose for a given component that it were composed of only 2 vertices (u and v). The only way this can be the case is if (u, v) is in only one or both of M_L and M_R. But both of these cases have already been covered in previous steps, and so any such component will not be present in $M_L \cup M_R$ at this point. That the component is essentially a straight-line with vertices alternating between L and R and edges alternating between M_L and M_R follows from the fact that all of these edges derive from original matchings M_L and M_R.

Choosing those edges that will result in a matching for the remaining vertices in L' and R' is straightforward if the number of vertices in the component is even (i.e. the dotted line in Figure 2 is not present):

starting at either end, we simply take every other edge. The result is that all vertices have exactly one edge incident upon them.

Note that it is possible for there to be a cycle in components with an even number of vertices. Furthermore, this cycle can only connect the tail to the head. Proof: In a chain of vertices such as those above, the only vertices with degree $\neq 2$ are the head and the tail. Hence, they are the only vertices to which another edge can be incident. These situations are not problematic, as taking every other edge will result in a matching over all of the vertices, no matter which edge we start with.

Figure 3: A cycle can arise in components with an even number of vertices.



The case that is slightly more complicated is when the number of vertices is odd. Note that, with an odd number of vertices, both the head and the tail are in L, or both are in R. WLOG, consider the case where they are both in L (essentially the same argument follows for the case where they are both in R). Note also that both the head and the tail cannot be in L'. Proof: suppose both the tail and head were in L'. Exactly one of them (call it u) has one incident edge upon it from M_R (since the edges alternate between M_L and M_R, and the number of edges is even), but that edge cannot also be in M_L since we would have removed it in a previous step. Hence, there is no edge from u that is in M_L, and by definition that M_L is a matching over L', u cannot be in L'. Our strategy for this case, then, is to not include the edge which connects the component to a vertex outside of L' (or R'), and then continue processing the now even-numbered set of vertices as we did above. Note: there can be no cycles in a component with an odd number of vertices, as this would connect the head and the tail which are either both in L or both in R, and there cannot exist any such edge in M_L or M_R.

2 Correctness

We will argue that the algorithm terminates and, once finished, returns a matching covering $L' \cup R'$, i.e. each vertex in $L' \cup R'$ is incident to exactly one edge of M.

The edges in $M_L \cup M'_R$ can be broken into three mutually exclusive categories, which correspond exactly to those which our algorithm describes:

1. Edges in both M_L and M_R
2. Edges which share no common vertices with any of the remaining edges
3. Edges which share common vertices with any of the remaining edges

That these groups are mutually exclusive follows immediately from their definitions: the second and third are defined as opposites of each other, and together they are defined over the remainder of the set from the first. Furthermore, we can think of these groups as corresponding to mutually exclusive sets of vertices in the original $L \cup R$, since by definition the first two groups are disconnected from the rest of the graph under M_L and M_R. Hence, as long as we can find a matching for the vertices in each of these groups individually, we will have found a matching for $L' \cup R'$ as well.

Indeed, that the algorithm finds a matching for those vertices in the first group has been justified in the description. That the vertices in the second group are covered under M also follows from the description. For the third group, in the case of a component with even vertices, that all of the vertices are covered under M is clear, since taking every other edge (whether there exists a cycle or not) will end up with each vertex having exactly one edge incident to it. In the case that there is an odd number of vertices in the component, we showed that both the head and the tail must be in either L or R, but both cannot be in L' or R', hence we can reduce the problem to case with an even number of vertices, for which the proof is the same as above.

3 Runtime

With an adjacency list of edges in M_L and M_R, the steps in the algorithm described are simple to complete. Note that, in the adjacency list, the number of vertices adjacenct to any given vertex is between 0 and 2, inclusive.

First, initialize the adjacency list based on the given matchings M_L and M_R. This can be done in $O(m) = O(n)$ time, since the number of edges is bounded by the number of vertices in $L' \cup R'$ as M_L and M_R are matchings. To determine those edges which share no common vertices (groups 1 and 2), simply traverse the list looking for vertices with exactly 1 adjacent vertex which in turn has no adjacent vertices except the vertex itself (i.e. for a given vertex u, if it has exactly one adjacent vertex v and v in turn has exactly one adjacent vertex u). This can be done in $O(n)$ time since each vertex must have a constant amount of work done for it (checking whether it has one neighbor and whether that neighbor has exactly one neighbor). Remove those edges which are found and add them to the matching M, and the remaining edges are exactly the third group.

The adjacency list can once again be iterated over in order to determine the structure of the remaining components as described above. Since each component contains at most 3 vertices, the number of remaining components is $O(n)$, and for each of them we must lookup whether the tail/head is in L' (or R'), which can be done e.g. using a binary search tree in $\log n$ time. From there, we do a constant amount of work to determine which edges should be included, as described above. Hence, the overall runtime is $O(n \log n)$ for the third step, and $O(n \log n)$ overall.