

Matching problems complete for logspace classes

Samir Datta

Rutgers University, New Brunswick, NJ, USA
sdatta@winlab.rutgers.edu

Sambuddha Roy

Rutgers University, New Brunswick, NJ, USA
samroy@paul.rutgers.edu

December 14, 2005

1 Introduction

It is well known that the counting versions of perfect matchings in bipartite graphs as well as general graphs are both $\#P$ -complete. On the other hand, no general reduction from matching in general graphs to bipartite matching is known. In a series of papers, authors have proven on the other hand, that counting the number of perfect matchings in a planar graph is in NC (while, intriguingly, finding a perfect matching in a planar graph is not known to be in NC). The current best known lower bound for general matching is NL. Given the state of affairs, one might be curious as to whether certain restricted matching problems are complete for NL. We show that that is indeed the case.

2 Preliminaries

There is considerable literature on matching theory (cf. [LovaszPlummer]). The principal open question for matchings is to prove that one can detect if a graph has a perfect matching or not in NC. As of now, we have [MVV] that matchings are only in RNC . We will show that the decision problem for matchings in graphs can be solved in Logspace for various classes of graphs. The strategy that we will follow for this will be starting with Tutte's theorem (where $oc(G)$ is the number of odd components in a graph G),

Theorem 1 *A graph G has a matching iff for $\forall S, S \subset G, oc(G \setminus S) \leq |S|$.*

Our viewpoint of the above will be that we will look at graphs for which it is sufficient to look at only small-sized S 's in order to decide whether G has a matching or not.

3 TreeMatching is complete for Logspace

TreeMatching is the decision problem where we are given an undirected tree (or in general we can allow a forest too), and the problem is to decide whether this has a matching or not.

We prove it is complete for Logspace by showing that the problem is in Logspace and then showing that it is hard for Logspace.

Proposition 2 *TreeMatching is in Logspace.*

Proof:(of Prop. 2) We have the following characterization for matchings in trees :

$$\begin{aligned} \text{tree } T \text{ has a perfect matching} \\ \text{iff} \\ \forall v \in T, oc(T \setminus v) = 1 \end{aligned}$$

one direction (\Rightarrow) follows from Tutte's theorem 1, while the other (\Leftarrow) follows by an easy induction on the size of the tree - since upon removal of a non-leaf node of the tree T , we are left with exactly one odd-component. By induction, we can thereby claim that a matching exists.

We can do all of this in Logspace because all we need to do is in parallel, for every $v \in T$, just check if there's exactly one odd component in $T \setminus v$. For trees, we can do Euler traversals in Logspace, and counting along an Euler traversal enables us to do the above. ■

Proposition 3 *TreeMatching is hard for Logspace.*

Proof:(of Prop. 3) For this direction we reduce the Logspace-complete problem, st-Connectivity in Directed Forests, to TreeMatching. A general way to reduce a connectivity question in a digraph to a matching question was given by [Karp, Upfal, Wigderson]. They reduce NL to Matching, in fact to BipartiteMatching. We describe their reduction in short : First, modify the graph G so that the source vertex s has a unique outgoing edge (call it e_s) and also vertex t has a unique incoming edge, e_t . In either bipartition of the constructed (bipartite) graph G' , keep a vertex for every edge in the digraph (call the two copies of an edge e $(e, 1)$ and $(e, 2)$). In G' put edges between every $(e, 1)$ and $(e, 2)$ if e is not incident with either s or t . If directed edges $e = (i, j)$ and $f = (j, k)$, then put an edge between $(e, 1)$ and $(f, 2)$. For the special edges e_s and e_t keep the vertices $(e_s, 1)$ and $(e_t, 2)$. Clearly the graph G' has a perfect matching iff there is a path from s to t in the digraph G . This completes [KUW]'s reduction.

Now, all we need to observe is that this reduction can be made to preserve topological properties : in particular, given graph G and a cut edge e in the underlying undirected graph, in the bipartite graph G' there is an edge corresponding to e namely the edge between $(e, 1)$ and $(e, 2)$. Thus, cut edges in G correspond to cut edges in G' - clearly then if G is a (directed) tree/forest, then G' is a (undirected) tree/forest.

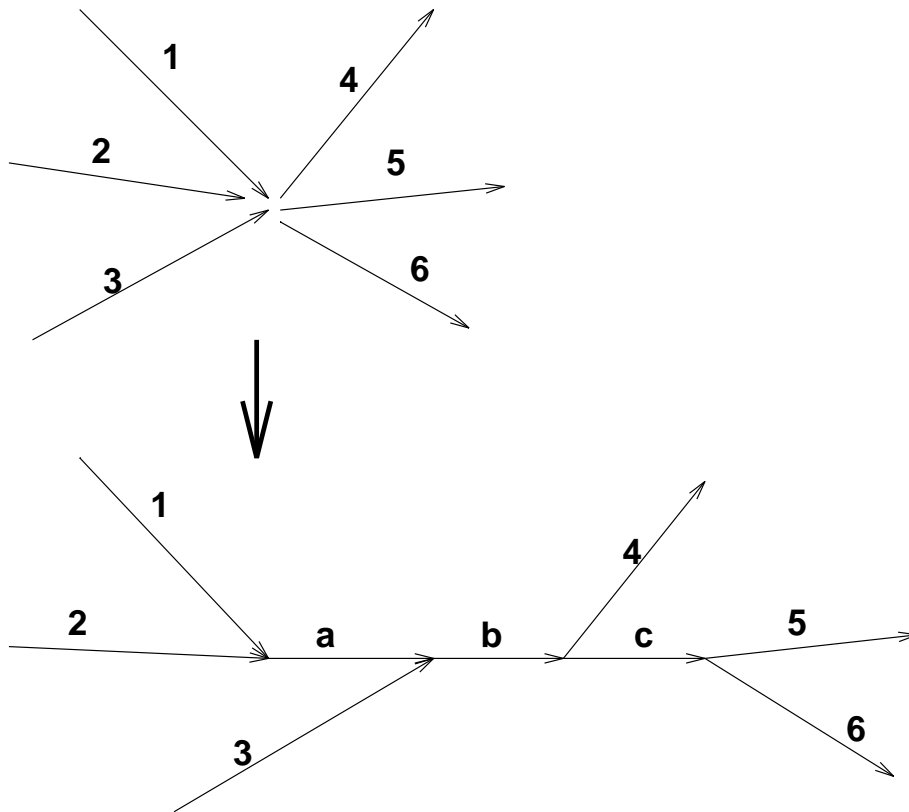


Figure 1: Situation at a vertex

Hence, TreeMatching is Logspace-hard. ■

We can in fact say something more : The [KUW] transformation can easily be made to preserve the genus of the underlying undirected graph of the given digraph G' . That is digraphs of a certain genus are reduced to bipartite graphs of the same genus. Therefore, for instance, st -Connectivity questions in planar digraphs are reducible to PlanarBipartiteMatching.

It is not exactly true that the [KUW] reduction transforms a planar digraph into a planar bipartite graph. For instance a planar digraph may have a vertex at which there are three incoming edges and three outgoing edges. Then if we label these edges $1, 2, \dots, 6$, in the constructed bipartite graph, we will have a $K_{3,3}$ between these vertices.

This is the only hurdle, and we can easily surmount this : whenever there is a vertex at which the number of incoming (n_i) and the number of outgoing (n_o) edges are ≥ 2 , then we extend the vertex out creating new directed edges as shown in the Fig. 1, or if the graph is not bimodal, we expand each vertex into a small directed cycle. This latter transformation introduces lots of cycles in our (originally maybe acyclic) digraph, but preserves connectivity, and every vertex has total degree (indegree + outdegree) at most three.

So, in either case, the connectivity questions in the old digraph clearly translate to those in the new one (and both the old and new digraphs have the same genus). Now we can convert the new digraph to a bipartite graph as per the [KUW] reduction, preserving genus.

4 A Matching problem complete for NL

Given a graph G , a near-perfect matching of the graph is a matching that covers every vertex of the graph but two. Now, consider the following (“promise”?) problem : Given a graph G with a near-perfect matching, labeling (some of) the edges in G , find out if G has a perfect matching or not.

Is this problem in NL? Yes, because all we have to do is to find an augmenting path and because the given matching M is near-perfect, we are assured that there exists exactly one augmenting path in the graph. Find out the two M -exposed vertices of the graph and then nondeterministically guess an M -augmenting path : for every node in the path which we reach through an M -exposed edge, we have no choice but to follow the matching of that node, and for every node which we reach through a matched edge, we nondeterministically guess which next node to go to via an M -exposed edge. Altogether if we can reach the other M -exposed vertex through these path augmentations, we have an augmenting path which thereby ensures a perfect matching of the entire graph.

The other direction again follows from [KUW] : if we start with a digraph and convert it into a (bipartite) matching question as via [KUW], the resulting bipartite graph always has a near-perfect matching.

Now, suppose a graph G is given along with a matching which falls 2 short of a perfect matching (compare with the above case where G always has a matching

which falls 1 short of perfect). In this case, is the detection of a perfect matching in NL still? Yes! Since the given matching falls two short of perfect, there are four vertices a_1, b_1, a_2, b_2 . Join b_1 and a_2 by a **matched** edge and ask if this modified graph has an augmenting path from a_1 to b_2 . Clearly the modified graph has an augmenting path iff there is a perfect matching in the original graph. Likewise, if we are given a graph G and a matching which falls k short of perfect ($k = O(1)$), then deciding whether the graph has a perfect matching is in NL. An open question is whether we can still decide the above in NL for k nonconstant.

5 Acknowledgement

We thank Eric Allender for insightful discussions and motivation.