Rainbow Perfect Matchings

October 8, 2012

Instructions

In this home assignment you are going to put together an algorithm yourself. In the previous labs you were given the complete algorithms. Here you will instead be presented with two pieces of the algorithm and it is up to you to assemble it. The idea is to mimic the behavior of an algorithm researcher/engineer. This is no doubt tougher than the labs and it is recommended that you read the whole home assignment and the related material listed in the end *several times* before you actually get to work.

Description

The Rainbow Perfect Matching problem (RPM) is the following:

You are given a multigraph $G = (V, E)$ with $|V| = 2n$, i.e. a graph in which there may be multiple copies of an edge u, v for any pair $u \neq v \in V$. In addition, each edge $e \in E$ has a color $c(e) \in [n]$. The question is whether there exists a *rainbow* perfect matching in G, i.e. a subset of the edge $E' \subseteq E$ such that

- Each vertex in V is the endpoint of exactly one edge in E'.
- There is exactly one edge in E' of every color.

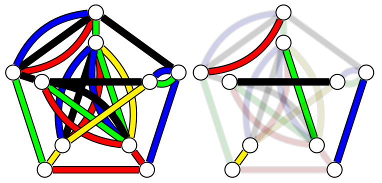
Figure 1: Left: An instance to the RPM problem. Right: A solution to the same instance.

Your task is to come up with an algorithm for a *restricted* version of RPM: We define bRPM as the RPM problem on *balanced bipartite* graphs, i.e. when we know that V can be partitioned in two equal halfs V_1 and V_2 such that every edge $e \in E$ has one endpoint in V_1 and the other in V_2.[1]

[1] Once you have found the algorithm for bRPM it is not too difficult to modify your algorithm to work for the general RPM. You'll need to exchange the biadjacency matrix for something else but similar. This is however not part of the assignment.

The bRPM problem is an NP-complete problem, which follows by an easy reduction from 3-Dimensional Perfect Matching ((8.20) in Kleinberg and Tardos). Thus, you should expect to end up with an exponential time algorithm. Below are the descriptions of the two pieces.

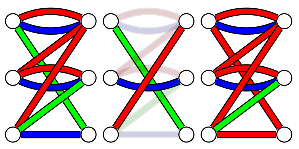
Figure 2: Left: A bipartite multigraph with coloured edges. Middle: A rainbow perfect matching. Right: A graph without a rainbow perfect matching.

Algorithmic Piece 1

bRPM generalizes ordinary bipartite graph perfect matching: Given a bipartite graph find out if there is a subset of the edges such that every vertex is part of precisely one of the chosen edges. In class (lecture 8, see also [S11] in the related material section at the end of this assignment) we learned how to solve for bipartite perfect matchings with an algebraic technique. We reiterate it here in a slightly modified variant without the use of fields of characteristic two. We anticipate that you probably will find the version below easier to implement:

The following algorithm decides if a bipartite graph G contains a perfect matching:

```
1. Fix a prime $p \gg n$.
2. In the biadjacency matrix $A_G$ of the graph $G$, replace each 1 with a random value between 0 and $p$. Make your random choices uniformly and independently at random.
3. Compute $m_G = \det(A_G) \pmod{p}$.
4. If $m_G \neq 0$ return "yes" else return "no".
```

The idea behind the algorithm is that m_G actually computes a polynomial in edge "variables", here replaced by random values, such that there is precisely one monomial in the polynomial for each perfect matching in the graph. The success guarantee is given directly by the Schwartz-Zippel lemma.

Algorithmic Piece 2

Suppose we had some efficient means to actually *count* the number of perfect matchings in a bipartite multigraph (no colors here), then we could use the technique of inclusion–exclusion (lecture 4, see also [Hu11] in the related material section at the end of this assignment) to count the rainbow perfect matchings in a bRPM instance. To see how, let $pm(G)$ for a bipartite (uncolored) multigraph G be the number of perfect matchings in G.

The following algorithm computes the number of rainbow perfect matchings in the input bRPM given by the graph G and a color function $c : E \to [n]$:

```
1. Set sum = 0.
2. For each $X \subseteq [n]$.
3.     Set $G_X = (V_1, V_2, E_X)$ where $E_X = \{e | e \in E, c(e) \in X\}$.
4.     sum = sum + $(-1)^{n-|X|} pm(G_X)$.
5. end
6. return sum.
```

The reason it works is that perfect matchings that do not use all colors, and hence necessarily use some color multiple times, will be counted an even number of times. Exactly half of them will be multiplied with the sign factor -1, so they will cancel each other. Formally, if the colors used by the perfect matching m is $C \subset [n]$, we will count the perfect matching m in $pm(G_X)$ for every $X \supseteq C$. For every such X for which $n - |X|$ is odd, we will subtract the matching. But for every X for which $n - |X|$ is even, we will add the matching. Altogether we will get a net result of zero from the matching since there are as many odd sized X's as even sized ones in the set $\{X | C \subseteq X \subseteq [n]\}$. Rainbow perfect matchings on the other hand, will only be counted once, in $pm(G_{[n]})$.

Unfortunately, we don't have any efficient algorithms to count the number of perfect matchings, but we *do* have algorithmic piece 1. We need to replace $pm(G)$ for something else that accounts for all perfect matchings in G. We will not be able to count the number of rainbow perfect matchings, but we will be able to detect if there is one.
Hint: the resulting algorithm should use one "variable" for each colored edge in the input bRPM above.

Deliverables

Algorithm description

Describe in pseudo code how the two algorithmic pieces can be combined into an algorithm for bRPM.

Runtime bound

Derive the runtime bound in $O(.)$ notation in terms of n.

Failure bound

Present an upper bound on the probability that the algorithm fails to report a solution to a bRPM instance that has a solution.

Implementation

Implement the algorithm in a programming language of your choice. The trickiest part is probably to compute the determinants. Note that you can *not* use Matlab's det and take the answer modulo the prime p since the internal calculations may overflow. You need to implement a determinant algorithm. In the appendix we present such an implementation of a determinant mod p function in Matlab. Use a prime p such that $p^2 < 2^{30}$. That way you can represent elements and count with them using a standard 32bit **int** datatype.

Benchmark

Construct some small instances of the bRBM and test your implementation. Use as large n as you seem fit with respect to the running time. The algorithm should finish in a few seconds for the largest n. Try to make interesting instances that have no solutions as well as instances in which you plant a solution. Test the behavior with respect to the choice of p. If you make the prime p just above n, say $\approx 2n$, you should be getting false negatives quite easily.

Related Material

[S11] http://www.cs.berkeley.edu/~sinclair/cs271/n2.pdf
[Hu11] http://arxiv.org/abs/1105.2942

Appendix

Matlab code to compute the determinant of a square matrix A modulo a prime p.

```
function [d]=detg(A,p)
  n=size(A,1);
  d=1;
  rs=zeros(1,n);
  for i=1:n,
    who=-1;
    for j=1:n,
      if rs(j)==0 && A(j,i)~=0,
        rs(j) = i;
        who = j;
        d = mod(d*A(j,i),p);
        break;
      end;
    end;
    if who==-1,
      d = 0;
      break;
    end;
    for j=1:n,
      if (rs(j)==0 && A(j,i)~=0)
        d=mod(d*modexp(A(who,i),p-2,p), p);
        A(j,:) = mod(A(who,i)*A(j,:)-A(j,i)*A(who,:),p);
      end;
    end;
  end;
  if (d>0),
    sgn = n;
    vis=zeros(1,n);
    for i=1:n,
      if vis(i)==0,
        j=i;
        sgn=sgn-1;
        while (vis(j)==0),
          vis(j) = 1;
          j=rs(j);
         end;
      end;
    end;
    if (mod(sgn,2)==1),
      d=p-d;
    end;
  end
end

function [r]=modexp(a,e,p)
  if (e==1)
    r = a;
  else
    r = modexp(a,floor(e/2),p);
    r = mod(r*r,p);
    if (mod(e,2)==1)
      r = mod(r*a,p);
    end
  end
end
```