CS 591-06, Lecture 4

Jared Saia
University of New Mexico

Class Outline

- Review of defn of "Augmented Chains"
- "Find-Matching" algorithm for finding matching in bipartite graph
- Proof of correctness for "augmented chains" algorithm
- Algorithm for making a graph regular (and exercise)
- The migration algorithms!

Matching Algorithm

For a bipartite graph G with edge set E:

Defn: A *chain* is a sequence of vertices and edges $v_1, e_1, v_2, e_2, ..., e_l, v_l$ such that the edge $e_i = (v_{i-1}, v_i)$.

Let M be some (possibly non-maximum) matching on G.

Defn: A chain $v_1, e_1, v_2, e_2, \ldots, e_l, v_l$ is called *M-alternating* if the edges $e_2, e_4, \ldots$ are in M.

Defn: A chain $v_1, e_1, v_2, e_2, \ldots, e_l, v_l$ is called *M-augmenting* if it is M-alternating and the vertices v_1 and v_l are not incident to any edges in M.

Chain Theorem: A Matching M is maximum for a graph G if and only if G contains no M-augmenting chains.

Algorithm Find-Matching

Input: A bipartite graph $G = (L, R)$

Algorithm *Find-Matching*:

1. Let $M = \phi$
2. While there are still unmatched vertices in L do
   (a) For each v that is unmatched in L do
      i. Search for an augmenting chain by doing a breadth first search from v
      ii. If an augmenting chain, A, is found, "augment" M by deleting from M all matched edges in A and by adding to M all unmatched edges in A.

Proof of Chain Theorem

Chain Theorem: A Matching M is maximum for a graph G if and only if G contains no M-augmenting chains.

Easy direction: If a matching M is maximum then G contains no M-augmenting chain.

To show this, assume, by way of contradiction, that M is a maximum matching and that there is an M-augmenting chain $C = v_1, e_1, v_2, e_2, v_3, e_3, v_4, e_4, \ldots, e_l, v_l$. Let $E' = M \cup \{e_2, e_4, \ldots, e_{l-1}\} - \{e_1, e_3, \ldots, e_l\}$.

Easy direction

We note the following:

- The edge set E' is also a matching on G, since each vertex v_i in the chain C has only one edge incident to it in E' and all other vertices not in C have at most one incident edge in E'(Why?)
- $|E'| > |M|$ since to create E', we added $\lceil l/2 \rceil$ edges and removed $\lfloor l/2 \rfloor$ edges (and l is odd).

This implies that E' is a larger matching than M which is a contradiction.

Hard Direction

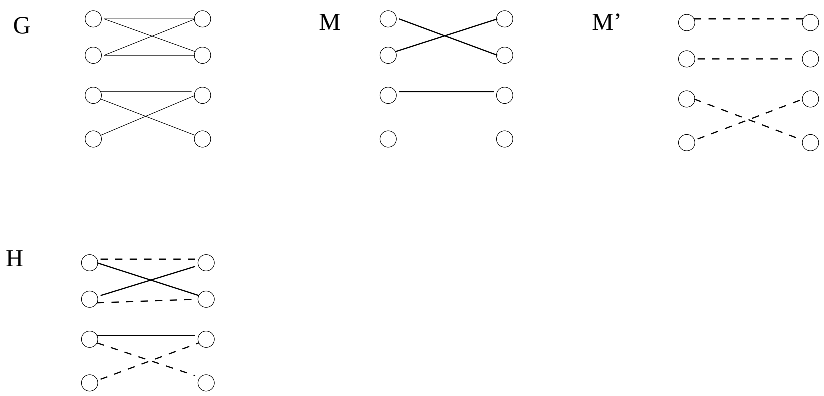
If G contains no M-augmenting chain, then the matching M is maximum.

Proof: Assume there exists some matching M' such that $|M'| > |M|$.

Let H be the subgraph of G which has only those edges which appear in either M or M' but not both. H is a graph of degree no more than 2 containing cycles and paths, the edges of which alternate between being in M and M'.

All cycles must be of even length (why?). Thus there must be some path with more M' edges than M edges. This path is an M-augmenting chain in G.

Picture of H



Tying up loose ends

We've shown that we can factor a k-regular graph into a bunch of 2-regular graphs, but what about the general case?

Recall that a demand graph is not necessarily a regular graph.

Q: Is there a way to make a graph regular with out increasing the maximum degree by much?

A: Yes!

Theorem: For any graph G with degree Δ, there is a graph G' with degree $\Delta' = 2\lceil \Delta/2 \rceil$ such that G is a subgraph of G'.

Making a graph Δ'-regular

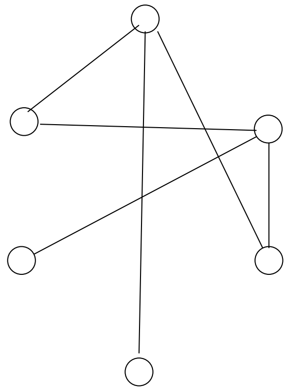
The following algorithm takes a graph with degree Δ and adds edges to it to get a graph of degree $\Delta' = 2\lceil \Delta/2 \rceil$

Algorithm *Regular*:

1. While there exists a vertex with degree less than $\Delta' - 1$, add a self loop to that vertex.
2. While there exist two distinct vertices of degree $\Delta' - 1$, add an arbitrarily directed edge between them.

Excercise

Make the following graph 4 regular as specified in Algorithm *Regular*



The algorithms

We can now present the entire algorithms for migration without space constraints.

This will complete Section 2 of the paper "On Algorithms for Efficient Data Migration"

Direct Migration

Input: A demand graph G with degree Δ

Output: A migration plan taking $3/2 \lceil \Delta/2 \rceil$ time steps.

Algorithm *Direct Migration*:

1. Make G $2 \lceil \Delta/2 \rceil$ regular using Algorithm *Regular*
2. Factor G into $\lceil \Delta/2 \rceil$ 2-factors using Algorithm *2-factor*
3. Take up to three time steps to send the objects associated with each 2-factor.

Indirect Migration

Input: A demand graph G with degree Δ and n vertices.

Output: A migration plan using $n/3$ bypass nodes and taking $2 \lceil \Delta/2 \rceil$ time steps.

Algorithm *Indirect Migration*:

1. Make G $2 \lceil \Delta/2 \rceil$ regular using Algorithm *Regular*
2. Factor G into $\lceil \Delta/2 \rceil$ 2-factors using Algorithm *2-factor*
3. Use $n/3$ bypass nodes to take *two* time steps to send the objects associated with each 2-factor.

Minute Paper

1. What was the most interesting thing you learned in the section on data migration?
2. What was the most confusing thing in the section on data migration?
3. What would you like to see more of? less of?