Application of Graph Matching

Jaehyun Koo (koosaga.com)

August 22, 2020

Introduction

We know how to find the Maximum Matching in $O(V^2E)$ time (Edm. 1965). Can we do better? What about related problems?

Introduction

We know how to find the Maximum Matching in $O(V^2E)$ time (Edm. 1965). Can we do better? What about related problems?

Matching can be done in $O(E\sqrt{V})$ time (Micali-Vazirani. 1980)

Weighted Matching in bipartite graph in $O(V^4)$ time (Kuhn. 1955)

Introduction

We know how to find the Maximum Matching in $O(V^2E)$ time (Edm. 1965). Can we do better? What about related problems?

Matching can be done in $O(E\sqrt{V})$ time (Micali-Vazirani. 1980)

Weighted Matching in bipartite graph in $O(V^4)$ time (Kuhn. 1955)

Weighted Matching in $O(V^4)$ with complex variant of Kuhn (Edm. 1965)

Weighted Matching in $O(V(E + V \log V))$ time (Gabow. 1990)

Weighted Matching in $O(E\sqrt{V} \times |WORD|)$ time *assuming a reasonable arithmetic model* (Duan, Pettie, Su. 2017)

Approximate (weighted) matching for dynamic graph is an active research topic, with different complexity and approximation ratio considered.

Introduction

We know how to find the Maximum Matching in $O(V^2E)$ time (Edm. 1965). Can we do better? What about related problems?

Matching can be done in $O(E\sqrt{V})$ time (Micali-Vazirani. 1980)

Weighted Matching in bipartite graph in $O(V^4)$ time (Kuhn. 1955)

Weighted Matching in $O(V^4)$ with complex variant of Kuhn (Edm. 1965)

Weighted Matching in $O(V(E + V \log V))$ time (Gabow. 1990)

Weighted Matching in $O(E\sqrt{V} \times |WORD|)$ time *assuming a reasonable arithmetic model* (Duan, Pettie, Su. 2017)

Approximate (weighted) matching for dynamic graph is an active research topic, with different complexity and approximation ratio considered.

Yes, this is very deep. Let's not get in further :)

Chinese Postperson Problem

Chinese Postperson Problem

Given a weighted graph, find a minimum-weight circuit that visits **every edges** at least once.

This resembles the *Traveling salesperson problem*, where we should visit **every vertices**.

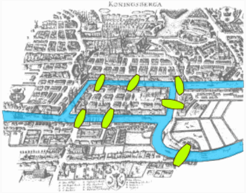
Chinese Postperson Problem

The obvious lower bound for the answer is $\sum_{e \in E(G)} w(e)$. This bound is achieved when there exists a circuit that visits all edges exactly once.

Chinese Postperson Problem

The obvious lower bound for the answer is $\sum_{e \in E(G)} w(e)$. This bound is achieved when there exists a circuit that visits all edges exactly once.

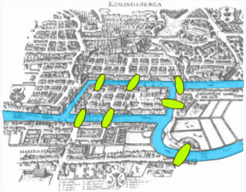
Reminds me of something...



Chinese Postperson Problem

The obvious lower bound for the answer is $\sum_{e \in E(G)} w(e)$. This bound is achieved when there exists a circuit that visits all edges exactly once.

Reminds me of something...



Theorem (Euler circuit). In a connected graph, there exists a circuit which visits all edges exactly once if and only if every vertex have even degree.

Proof by induction on $|E(G)|$ is somewhat well-known.

Chinese Postperson Problem

Graph with an Euler circuit can be described in a simple way, which means it is desired.

Our goal of finding a circuit that visits every edge *at least once* is somewhat cumbersome. Can we stick to the notion of Euler circuit?

Chinese Postperson Problem

Graph with an Euler circuit can be described in a simple way, which means it is desired.

Our goal of finding a circuit that visits every edge *at least once* is somewhat cumbersome. Can we stick to the notion of Euler circuit?

Change the goal! Rather than finding a circuit, *duplicate* the edges to make the graph to have a Euler circuit.

We should **minimize** the total weight of the augmented graph.

Chinese Postperson Problem

Given a graph, duplicate the edges to make the graph Eulerian, while minimizing the total edge weight.

After the duplication, every edge should have even degree.

There are even number of odd degree vertices. Suppose there are two of them. Can we solve it?

Chinese Postperson Problem

Given a graph, duplicate the edges to make the graph Eulerian, while minimizing the total edge weight.

After the duplication, every edge should have even degree.

There are even number of odd degree vertices. Suppose there are two of them. Can we solve it?

We can duplicate all edges in shortest path.

We can't do better: Set of duplicated edges should connect two odd-degree vertices.

Case of two odds: Find the shortest path between two, and duplicate along the path.

Chinese Postperson Problem

What about the *case of four odds*?

There are three ways to match four of them into two pairs. Can't we try all of matches, and do the same for each pairs?

Chinese Postperson Problem

What about the *case of four odds*?

There are three ways to match four of them into two pairs. Can't we try all of matches, and do the same for each pairs?

In general, this works in matching framework. For each pair of odd-degree vertices, we can assign the weight as a length of shortest path, and find the optimal matching.

Using Floyd-Warshall and algorithm from Gabow gives $O(N^3)$ solution.

This certainly gives the solution, but is this optimal?

Chinese Postperson Problem

We surely don't have to duplicate same edge two times.

Consider the set of duplicated edges from optimal solution.

This solution is acyclic. Otherwise, remove that cycle.

Chinese Postperson Problem

We surely don't have to duplicate same edge two times.

Consider the set of duplicated edges from optimal solution.

This solution is acyclic. Otherwise, remove that cycle.

Take any nontrivial component of it. It should have two *leaf* vertices (degree 1).

Take the unique path, remove it, and repeat it.

Now we decomposed this set into the disjoint path between some pair of odd-degree vertices. Our solution can't be worser than that.

Christofide’s Algorithm

Christofide's Algorithm

Although the Chinese postperson problem was solved in polynomial-time, it's vertex variant, the *Traveling salesperson problem* can't be solved.

But we can try to **approximate**: Can we find a solution that is close to the optimal solution?

*W*hat is *close enough* solution? We use the notion that the algorithm returns a solution that is always not worser than the optimal solution by some multiplicative factor.

For example, we will talk about an algorithm that returns a solution that is **never worser** than 1.5 times the optimal solution.

But it only works in a special case: The distance should form a *metric space*.

Christofide's Algorithm

In *traveling salesperson problem*, we should visit every vertices and minimize the distance of total tour.

Let's try to make a following assumption on the *distance between two points*.

1. $d(x, y) = 0$ iff $x = y$
2. $d(x, y) = d(y, x)$
3. $d(x, y) + d(y, z) \geq d(x, z)$

Corollary. If those three axioms holds, then $d(x, y) \geq 0$.

A distance function is a *metric*, if those three axioms hold. Those assumption hold on $\mathbb{R}^n$, shortest path in graph, etc.

Christofide's algorithm requires the distance function to be a metric.

Christofide's Algorithm

The *minimum spanning tree* of the connected graph is the connected subset of edges that have minimum weight.

It is the minimum weight base of graphic matroid.

Christofide's Algorithm

The *minimum spanning tree* of the connected graph is the connected subset of edges that have minimum weight.

It is the minimum weight base of graphic matroid.

TSP tour is a cycle, so it's connected in the end. Thus the length of TSP tour is at least the size of MST.

Christofide's Algorithm

The *minimum spanning tree* of the connected graph is the connected subset of edges that have minimum weight.

It is the minimum weight base of graphic matroid.

TSP tour is a cycle, so it's connected in the end. Thus the length of TSP tour is at least the size of MST.

Let's duplicate all edge of MST, and do a *Euler tour* on the graph. It visits all the vertices, possibly more than once.

If a vertex is visited more than once, simply remove it from cycle (We use axiom 3 of metric space)

Christofide's Algorithm

The *minimum spanning tree* of the connected graph is the connected subset of edges that have minimum weight.

It is the minimum weight base of graphic matroid.

TSP tour is a cycle, so it's connected in the end. Thus the length of TSP tour is at least the size of MST.

Let's duplicate all edge of MST, and do a *Euler tour* on the graph. It visits all the vertices, possibly more than once.

If a vertex is visited more than once, simply remove it from cycle (We use axiom 3 of metric space)

We obtained a TSP tour which have at most $2 \times$ (MST weight) $\leq 2 \times OPT$ (OPT denotes size of optimal solution).

In other words, this is a **2-approximation** algorithm for metric TSP.

Christofide's Algorithm

So, we have a MST, we want an Euler tour, so we duplicated the edge..
But we've talked tirelessly about this stuff!

Rather than naively duplicating, let's gather all odd-degree vertices in MST and find a minimum weight matching.

Add the matching to the tree. The graph is Eulerian. Do the same thing. Much better, but how much?

Christofide's Algorithm

So, we have a MST, we want an Euler tour, so we duplicated the edge..
But we've talked tirelessly about this stuff!

Rather than naively duplicating, let's gather all odd-degree vertices in MST and find a minimum weight matching.

Add the matching to the tree. The graph is Eulerian. Do the same thing. Much better, but how much?

Lemma. The added min-weight matching has weight at most $0.5 \times OPT$

Proof. Take a TSP with weight OPT. Pull out all even-degreed vertex (By axiom 3 this decreases the length). Now you have a tour that visits all odd-degree vertices and only that.

Now, take the even-indexed edges, and take the odd-indexed edges. Both are a perfect matching: If the lemma don't hold, their total length will be greater than OPT.

Christofide's Algorithm

So, we have a MST, we want an Euler tour, so we duplicated the edge..
But we've talked tirelessly about this stuff!

Rather than naively duplicating, let's gather all odd-degree vertices in MST and find a minimum weight matching.

Add the matching to the tree. The graph is Eulerian. Do the same thing.
Much better, but how much?

Christofide's Algorithm

So, we have a MST, we want an Euler tour, so we duplicated the edge..
But we've talked tirelessly about this stuff!

Rather than naively duplicating, let's gather all odd-degree vertices in MST and find a minimum weight matching.

Add the matching to the tree. The graph is Eulerian. Do the same thing. Much better, but how much?

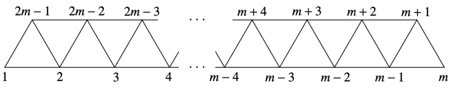
Lemma. The added min-weight matching has weight at most $0.5 \times OPT$

Proof. Take a TSP with weight OPT. Pull out all even-degreed vertex (By axiom 3 this decreases the length). Now you have a tour that visits all odd-degree vertices and only that.

Now, take the even-indexed edges, and take the odd-indexed edges. Both are a perfect matching: If the lemma don't hold, their total length will be greater than OPT.

Christofide's Algorithm

One of the worst case (where 1.5 is achieved) looks like the following:



Round trip through the faces uses $2m-1$ edges.

However, if your MST is unluckily a path from $1, 2, \dots, m$, along with all edges $i, 2m-i$, then the matching will spend extra $m-1$ edges.

If you think that's too unlucky, perturb slightly to make it a unique MST.

Parity shortest paths

You are given a positive-weighted undirected graph.

A problem of finding a shortest walk between two vertices s, t are already considered. Shortest walk are necessarily simple (So, they are shortest path).

Finding a shortest *odd* or *even* walk can be done similar. By odd/even, it concerns about the parity of number of edges.

Parity shortest paths

You are given a positive-weighted undirected graph.

A problem of finding a shortest walk between two vertices s, t are already considered. Shortest walk are necessarily simple (So, they are shortest path).

Finding a shortest *odd* or *even* walk can be done similar. By odd/even, it concerns about the parity of number of edges.

Create a copy of graph G, G', and for all edge $e = \{u, v\}$, connect $G(u) \leftrightarrow G'(v)$, $G(v) \leftrightarrow G'(u)$.

Then, any path between G, G' have odd length, and between themselves have even length. So you can find a path between $G(s), G(t)$ or $G(s), G'(t)$.

Are they simple walks in reality?

Parity shortest paths

Why should I care? Actually I'm not sure, but we can agree that they are pretty fundamental questions to ask in graphs.

For example, if you can find a shortest even/odd path between s, t, then you can find a shortest even/odd cycle in an undirected graph, by enumerating all $e = \{u, v\}$ and finding a shortest parity path between u, v in $G - e$.

You know, odd cycles define bipartite graphs.

Parity shortest paths

Why should I care? Actually I'm not sure, but we can agree that they are pretty fundamental questions to ask in graphs.

For example, if you can find a shortest even/odd path between s, t, then you can find a shortest even/odd cycle in an undirected graph, by enumerating all $e = \{u, v\}$ and finding a shortest parity path between u, v in $G - e$.

You know, odd cycles define bipartite graphs.

What about directed graph? You can't even find the **existance** of parity path in polynomial time (Thomassen 1985).

But, shortest odd cycle in directed graph are easy to find.

Existance of even cycle can be found in polynomial time, using some fancy algebra, done by fancy researchers not long ago (Robertson, Seymour, Thomas 1999)

Parity shortest paths

This trick is very simple and beautiful: One of my favorite solutions ever.

But I don't know how to describe the intuition.

Parity shortest paths

This trick is very simple and beautiful: One of my favorite solutions ever. But I don't know how to describe the intuition.

WLOG, assume we are finding even length path between $s \neq t \in V(G)$.

Create a copy of graph G, G'. For all edge $e = \{u, v\}$, connect $G(u) \leftrightarrow G(v)$, $G'(u) \leftrightarrow G'(v)$.

Also, connect $G(v) \leftrightarrow G'(v)$ for all $v \in V(G)$.

If you find a perfect matching, you will probably end up with some vertices connecting itself, and others covered in even-length cycles.

Parity shortest paths

This trick is very simple and beautiful: One of my favorite solutions ever. But I don't know how to describe the intuition.

WLOG, assume we are finding even length path between $s \neq t \in V(G)$.

Create a copy of graph G, G'. For all edge $e = \{u, v\}$, connect $G(u) \leftrightarrow G(v)$, $G'(u) \leftrightarrow G'(v)$.

Also, connect $G(v) \leftrightarrow G'(v)$ for all $v \in V(G)$.

If you find a perfect matching, you will probably end up with some vertices connecting itself, and others covered in even-length cycles.

Let's erase the vertex $G(s), G'(t)$.

Parity shortest paths

Let's erase the vertex $G(s), G'(t)$.

Then, if a perfect matching exists, s and t is connected by an even-length path. We can see the existance.

Why is it true? Note that each edges are selected at most once. Otherwise, we discard both and connect $G(v), G'(v)$ instead.

Among the selected edges, there exists two odd-degree vertices. They should belong in same connected component, which is a path (degree is at most 2!)

Parity shortest paths

Let's erase the vertex $G(s), G'(t)$.

Then, if a perfect matching exists, s and t is connected by an even-length path. We can see the existance.

Why is it true? Note that each edges are selected at most once. Otherwise, we discard both and connect $G(v), G'(v)$ instead.

Among the selected edges, there exists two odd-degree vertices. They should belong in same connected component, which is a path (degree is at most 2!)

Generalize with weights: Give the edges their weight, and connect $G(v) \leftrightarrow G'(v)$ for all $v \in V(G)$ with cost 0.

Minimum weight perfect matching gives exactly the shortest path. Cycles doesn't exist because they are not cost-efficient.