CS491G Combinatorial Optimization
Lecture Notes

Jingjing Chen
May 23, 25 2001 –Summer

1 Introduction

A graph $G=(V, E)$ consists of vertex-set V and edge-set E,we may write $e=vw$ to indicate that the ends of e are v and w. A subgraph H of G is a graph such that $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$,and each $e \in E(H)$ has the same ends in H as in G. A path P in a graph G is a sequence $v_0,e_1,v_1,e_2 \ldots\ldots e_k, v_k$ where each v_i is a vertex, each e_i is an edge, and for $1 \leqq i \leqq k$,the ends of e_i are v_{i-1} and v_i. It is a circuit if it is closed when $v_0 = v_k$.The graph G is connected if every pair of vertices is joined by a path.

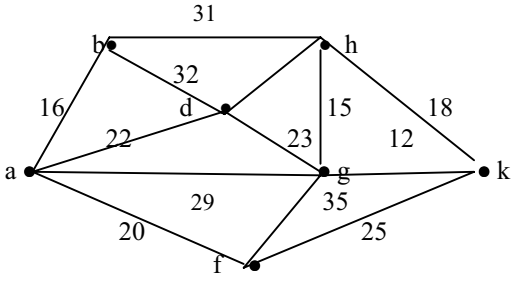
Definition:1.1 A connected graph with no circuit is called ***tree*** .
Definition:1.2 A subgraph H of G is a ***spanning tree*** if $V(H)=V(G)$.

2 Minimum Spanning Tree Problem

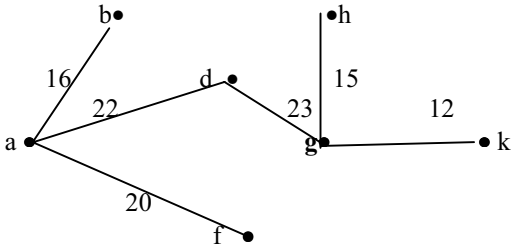
MST(Minimum Spanning Tree Problem):

> Given a connected graph $G = <V, E>$ and a real cost c_e for each $e \in E$, find a minimum cost spanning tree of G.

Consider the following graph with given costs Figure (a) ,we can find the MST from this graph as Figure (b).



(a)



(b)

How can we figure out MST easily? We will present some algorithms later; here we give some elementary results of graph theory:

Lemma 2.1: Every spanning tree on 2 or more vertices has at least one vertex of degree 1

Proof: By way of contradiction: Let ***G*=(*V,E*)** be a tree with the minimum degree 2; we show that it must contain a cycle. Starting with an arbitrary vertex say *a,* construct a path to the next adjacent vertex (say *b*) and from *b* to the next adjacent vertex and so on. Since each vertex has degree at least 2, we can do this at least *n* times, i.e., we have a path of length *n.* However, any path having n edges, must contain at least one cycle and hence ***G*** cannot be a tree. It follows that at least one vertex must have degree 1. (See Figure 2.1)

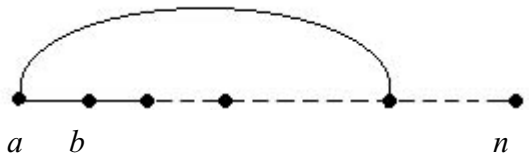


Figure2.1

Lemma 2.2: Every spanning tree has exactly n-1 edges

Proof: We argue it by induction on vertices.
(1) Base case: if there are 2 vertices in the tree, it must have exactly one edge, shown as Figure 2.2:

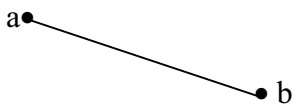


Figure 2.2

(2) Assume the Lemma is true for all trees with vertices $\leq$ n-1.
(3) Show it is true for any tree with n vertices.

a) By Lemma 2.1, a tree must contain at least one vertex of degree 1, so let us delete that degree 1 vertex.

b) Now the tree contains n-1 vertices; By inductive hypothesis we know that a tree of n-1 vertices has exactly n-2 edges.

Put the vertex back into the tree (just added only one edge for that is degree 1vertex), so the tree with n vertices should have (n-2)+1=n-1 edges, done.

3 Algorithms for the MST problem

3.1 The cut theorem & Prim's Algorithm

Definition 3.1.1: Given $G=<V\ E>$, $S \subseteq V$, the set $\delta(S)= \{vw \mid v \in S, w \notin S\}$ is called a ***cut or cut-set.***

Example of a cut:

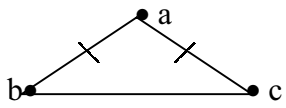


Figure3.1

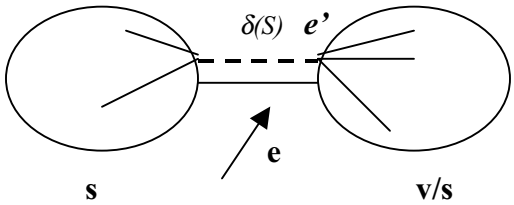
From Figure 3.1, $\delta(S)$= { e_{ab} , e_{ac} }is a cut and S={a}, V/S={ b, c } .

Proposition: If we add an edge into a tree, it creates exactly one cycle, then we remove any edge of that cycle ,it recreates a tree.

Lemma 3.1.2: Blue Rule

The lightest weight edge of any cut must be part of the MST.

Proof:



the lightest edge

Figure 3.2

Consider a cut $\delta(S)$ such that the minimum weight edge of $\delta(S)$ (say e) is not part of any MST. Clearly this cut is connected in the MST by some edge say e', where $w(e) \leq w(e')$. Throw e into the tree; from above proposition, we know it must create a cycle. Remove e' from the cycle recreating a tree of weight at most the weight of the original tree.

The implementation of Prim's Algorithm for MST

- Start with T={r} (r be any vertex of the T),
- Add the lightest weight cut edge to T,
- Stop when you have a connected tree.

Example:

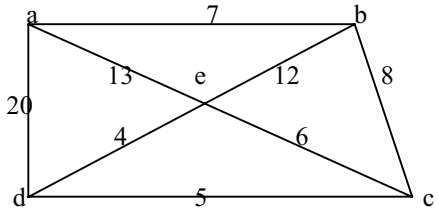
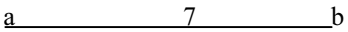
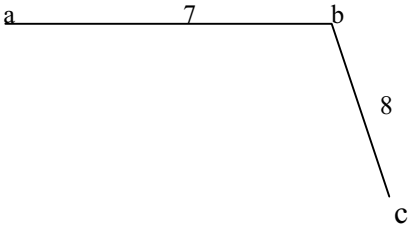


Figure 3.3

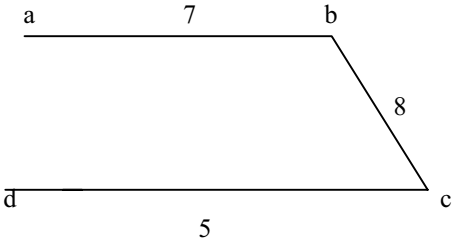
Applying Prim's Algorithm to Figure 3.3,
Start with vertex *a*: the lightest weight edge is *ab* (7<13<20), so add *ab* into *T*.



For *b*: the lightest weight edge is *bc* (8<12), add *bc* into *T*.



And go on, for *c*(5<6), we can add *cd* into *T*,



for *d*(4<20), *de* is added into *T*, the following Figure 3.4 is just *MST*.

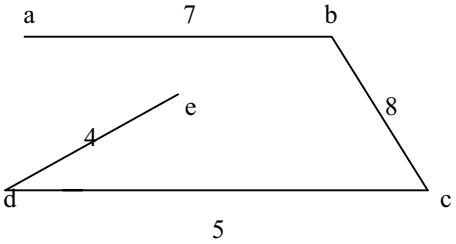


Figure 3.4

3.2 The cycle theorem & Kruskal's Algorithm

Lemma 3.2.1 Red Rule

Given any cycle in a graph, the heaviest edge cannot be part of any *MST*.

Proof:

Let us consider a cycle *C* with the heaviest edge *e*, such that *e* is part of some *MST*, as shown in Figure 3.5 :

Claim: There is at least one-edge of C, which creates a cycle with *e* in *T*.

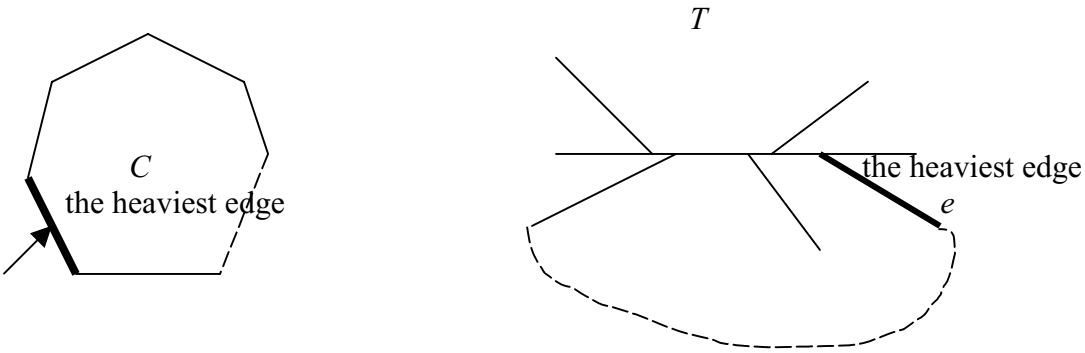


Figure 3.5

Throw the edges of the cycle *C* into the *T* one by one.

If no edge of the cycle creates a cycle with *e,* then we cannot find *C* in the graph either (since all the edges of the cycle have been added without creating the original cycle). Thus, there must exist at least one cycle with *e*; deleting *e* from the cycle creates a cycle of weight at most the weight of the original tree.

The implementation of Kruskal's Algorithm for MST

- Order the edges by weight (such as $e_1 \le e_2 \le e_3 \le \ldots\ldots\ldots \le e_m$),
- Throw each edge into the *MST* as long as it does not create a cycle.

Take Figure3.3 For example:

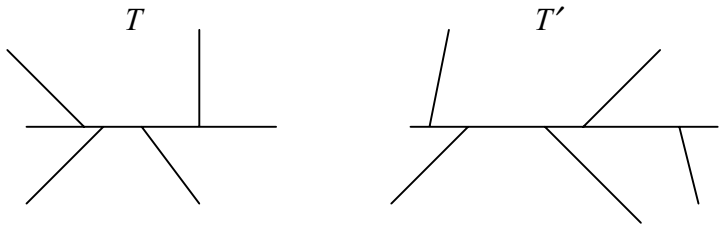
First We order the edges by weight:

de, dc, ce, ab, bc, be, ae, ad.

And then add de, dc, ab, bc into the *T*,ensure that these edges cannot create cycle.

We get *MST* as shown in Figure 3.4

3.2.2 Greedy proof of Kruskal's Algorithm



- Suppose *T* produced by Kruskal's Algorithm, the edges ordered by weight, such as $e_1 \le e_2 \le e_3 \le \ldots e_k \ldots \le e_m$. e_1 must be part of any *MST*.
- Let T′ be the MST produced by some other Algorithm, the ordered edges by weight, such as $e_1' \le e_2' \le \ldots \ldots e_k' \ldots \le e_m'$. We must have $e_1 = e_1'$; otherwise e_1 can be thrown into the non-Kruskal tree and any edge other than e_1 can be deleted from the resulting cycle to create a tree of weight at most the weight of the original tree.
- Let *k* be the first index such that $e_k' \neq e_k$. Clearly, $e_k' \geq e_k$; or else e_k' would have been considered first and hence would be in *T*.
- Push e_k into *T′,* a cycle is created but only with edges $e_k' \ldots e_m'$ .This is because, if a cycle was created with edges $e_1 \ldots e_k$, then that cycle would exist in *T* as well.
- Keep inserting edges of *T* into *T'* and deleting non-*T* edges as they are created; finally *T* is created from *T*' and we are done.

4 Basis Extension Theorem

Definition 4.1 A set of edges $B \subseteq E$, where $G = < V, E >$ is called **MST-*extensible***, if $B \subseteq T$ for some *MST* .

Theorem 4.2 Let *B* is a MST- extensible set, let *D* be a cut-set such that $B \cap D = \phi$, and let *e* be a minimum weight edge of *D*, then $B \cup \{e\}$ is MST-*extensible.*

Proof:

♦ *B* is MST-*extensible*, so $B \subseteq T$ (some *MST*),

(a) If $\{e\} \subseteq T$, we are done.

(b) Consider the case that $\{e\}$ is not $\subseteq T$. The tree *T* must contain at least one edge of the cut-set D say $f \neq e$; if not, it is not connected. Adding $\{e\}$ to *T*, creates a cycle with *f*. Since *e* is the lightest edge of the cut-set D, throwing *f* out recreates a tree of weight at most the original weight.

References

[1] William Cook, William H.Cunningham, William Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. John Wiley & Sons ,1998.