

Combinatorial Optimisation

HT 2008

Chapter 1

Minimum spanning trees

Some graph definitions

A *graph* $G = (V, E)$ consist of a set V of *nodes* or *vertices*, and a set E of unordered pairs of distinct nodes, the *edges*. When $\{u, v\}$ is an edge we say that the nodes u and v are *adjacent* and may write $u \sim v$.

A *walk* is a sequence of $k \geq 0$ nodes v_0, v_1, \dots, v_k that satisfies $v_i \sim v_{i+1}$ for each $i = 0, \dots, k-1$.

A *path* is a walk without repeated nodes (that is, $i \neq j \Rightarrow v_i \neq v_j$).

A *closed walk* is a walk that satisfies $v_0 = v_k$.

A *cycle* is a closed walk with $k \geq 3$ vertices and which has no repeated vertices other than $v_0 = v_k$.

A graph G is *connected* if there is a path between u and v for all (distinct) vertices u and v in V .

Graphs (continued)

If G is connected and *acyclic* (does not have cycles) then we say it is a *tree*.

A *subgraph* of G is a graph $G' = (V', E')$ with $V' \subseteq V, E' \subseteq E$. The subgraph G' is a *spanning subgraph* if $V' = V$.

A *spanning tree* in a connected graph G is a spanning subgraph which is a tree.

A *connected component* of G is a maximal connected subgraph. The number of components of G is denoted by $\kappa(G)$.

We will often identify subgraphs, walks, paths, cycles, etc. with their edge sets.

Minimum spanning trees

Let $G = (V, E)$ be a connected graph with $|V| = n$ nodes and $|E| = m$ edges, such that each edge e has a positive cost $c(e)$. We must find a set $E' \subseteq E$ of edges such that the graph $G' = (V, E')$ is connected and, subject to this, the total cost of E' is minimal.

Thus we seek a minimal cost spanning tree T of G .

Kruskal's greedy algorithm

input the connected graph G

initialise T to be \emptyset

while edges remain in G

 delete a cheapest edge e

if $T \cup \{e\}$ is acyclic **then** add e to T

return T

Theorem 1 Let G be a connected graph. Then Kruskal's algorithm yields a minimal cost spanning tree T .

Preliminary results

(a) Let G be an acyclic graph with n nodes. If G has m edges then $\kappa(G) = n - m$.

For, this is true if $m = 0$. Let $m \geq 1$, and suppose the result is true for $m - 1$. Consider an acyclic graph G with m edges. Form G' by deleting an edge e . By the induction hypothesis $\kappa(G') = n - m + 1$. But e must join two different components of G' (since otherwise there would be a cycle), and so $\kappa(G) = \kappa(G') - 1 = n - m$, as required.

(b) A graph with n nodes is a tree if and only if it is acyclic and has exactly $n - 1$ edges.

For, if G is acyclic and has m edges then by (a) we have $\kappa(G) = n - m$, so G is connected (that is, $\kappa(G) = 1$) if and only if $m = n - 1$.

(c) Let G be a graph with n nodes and edges set E . Let T be a spanning tree in G and let $e \in E \setminus T$. Then $T \cup \{e\}$ contains a unique cycle C ; and if f is any edge in C then $(T \cup \{e\}) \setminus \{f\}$ is a spanning tree.

If $e = \{u, v\}$ then there is a unique path in T between u and v , and so $T \cup \{e\}$ contains a unique cycle C . But then the graph with edges $(T \cup \{e\}) \setminus \{f\}$ is acyclic and has $n - 1$ edges, so it is a spanning tree.

Proof of theorem 1

1. T is a spanning tree.

For we never allow a cycle to be formed so T is acyclic; and if it were not connected then some edge $e \in E$ would join two connected components of T , and e should have been added to T .

We must show that T is optimal.

2. Let e_1, e_2, \dots, e_{n-1} be the (ordered) list of edges chosen to add to T . Let T^* be an optimal tree with $|T \cap T^*|$ as large as possible. Suppose that $T^* \neq T$. We shall deduce a contradiction, thus completing the proof.
3. Since $T^* \neq T$, for some j with $0 \leq j < n-1$ each of e_1, \dots, e_j is in T^* but e_{j+1} is not. By the lemma, since e_{j+1} is not in the tree T^* , there is a unique cycle C in $T^* \cup \{e_{j+1}\}$.

4. Note that C is not contained in T . Pick $f \in C \setminus T$. The edges e_1, \dots, e_j, f are all in T^* (and are all distinct), and so they do not contain a cycle. But after choosing e_1, \dots, e_j the algorithm chose e_{j+1} not f next. Hence $c(e_{j+1}) \leq c(f)$.
5. By the lemma, $\hat{T} = (T^* \cup \{e_{j+1}\}) \setminus \{f\}$ is a spanning tree. Also \hat{T} has length at most that of the optimal spanning tree T^* and so \hat{T} is also optimal. But $|T \cap \hat{T}| = |T \cap T^*| + 1$, contradicting our choice of T^* .

□

Time complexity

If we implement Kruskal's greedy algorithm on a computer, how long will it take to find a minimum spanning tree? We are especially interested in the time needed as the size of the input (G together with the costs c) grows large.

The time needed will of course depend on this size, but also on the type of machine used. We will count the number of 'steps' or 'elementary operations' (such as addition, multiplication) the algorithm uses as a function of the input size.

We may think of the graph as being given as follows: for each node u we are given a list of the adjacent nodes v together with the cost $c(e)$ of the edge $e = (\{u, v\})$.

There is no real point in distinguishing between n^2 and $3n^2$ steps.

Order symbols

Let $f, g : \mathbb{N} \rightarrow (0, \infty)$ be functions.

$f(n) = O(g(n))$ if there exists c such that $f(n) \leq cg(n)$ for n sufficiently large (or equivalently, if $\limsup_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$).

$f(n) = o(g(n))$ if $\frac{f(n)}{g(n)} \rightarrow 0$ as $n \rightarrow \infty$.

$f(n) = \Omega(g(n))$ if there exists $c > 0$ such that $f(n) \geq cg(n)$ for n sufficiently large (or equivalently, if $g(n) = O(f(n))$).

$f(n) = \Theta(g(n))$ if both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

For example: $n^2 = o(2^n)$ and $3n^2 = \Theta(n^2 + n)$.

Time complexity for Kruskal's greedy algorithm

We can sort the m edges in time $O(m \log m)$.

In fact we can implement the method so that the total time is also $O(m \log m)$.

The trick is to test quickly if $T \cup \{e\}$ is acyclic. To do this we may maintain a data structure which tells us for each node which component of the forest T so far chosen contains the node. If the end nodes of a possible new edge e are in different components then the edge e is added to T and the components merged.

Other algorithms to find a MST

There are other ways of organising the basic greedy strategy.

For example, in the method of Jarník (or Prim), the edge set T chosen always forms a tree, and we add a cheapest edge between the tree and the rest of the graph (which cannot introduce a cycle). Thus we start from an arbitrary single node v_1 , add a cheapest edge incident with v_1 , say the edge $\{v_1, v_2\}$, add a cheapest edge between v_1 or v_2 and the rest of the graph, and so on.

For further details on these algorithms (and others, for example that of Boruvka which is well suited to parallel computation), see for example Brassard and Bratley *Algorithmics* or Ahuja, Magnanti and Orlin, *Network Flows*.

General approach

Here is a general approach that covers the above methods and others.

A *cut* is the set of edges between B and $V \setminus B$ for some non-empty set $B \subsetneq V$ of nodes.

Red-green colouring

On input the connected graph G with costs on the edges, colour the edges green (in) or red (out) by applying the following rules, in any order. The 'green rule' picks edges, and the 'red rule' discards edges.

Green rule: find a cut containing no green edge, and a minimum cost uncoloured edge in the cut, and colour the edge green.

Red rule: find a cycle containing no red edge, and a maximum cost uncoloured edge in the cycle, and colour the edge red.

Prim's method consists of repeatedly applying the green rule, with the cut being the set of edges between the current tree and the rest of the graph.

What about Kruskal's method? Let the edge $e = \{u, v\}$ be uncoloured and be a cheapest such edge. If there is a path of green edges between u and v then we can colour e red; and if not, then we can colour e green – let B be the set of nodes joined to u by a path of green edges, and let X be the corresponding cut.

This shows also that, however the rules have been applied so far, if some edge is still uncoloured then we can colour another edge; and thus any partial colouring can be completed following the rules.

The 'red-green' Theorem

However the rules are applied, there is a minimum spanning tree which contains all the green edges and none of the red ones.

Corollary

Both Kruskal's and Prim's algorithms return a MST.