Lecture 9: Combinatorial optimisation

Anders Johansson

2011-10-22 lör

Contents

1 Combinatorial optimisation

Given a finite set $S \subset \{0,1\}^E$, sequences of 0 and 1 idexed by elements in some finite set E. A typical combinatorial optimisation problem is to find the minimum (or maximum) of some *objective function* $f(x)$, where $x \in S$.

Shortest path (SP) problem If S is the set of st-paths in a graph $G = (V, E)$ and $\ell : E \to R_+$ is a prescribed positive length. Minimise

$$\ell(x) = \sum_{e \in x} \ell(e) = \sum_{e \in E} \ell(e)x(e),$$

where in the last expression we consider a path x as a vector

$$x = (x_{e_1}, \dots, x_{e_m}) \in \{0,1\}^E.$$

Minimum spanning tree (MST) problem Let $w : E \to R$ be a given weighting of the edges in a graph $G = (V, E)$. Let S be the set of spanning trees and minimise $w(T) = \sum_{e \in T} w(e)$.

The Huffman coding problem For

$$S = \{ \text{ complete binary trees with leaf weights } w(1), \dots, w(m) > 0\}$$

minimise

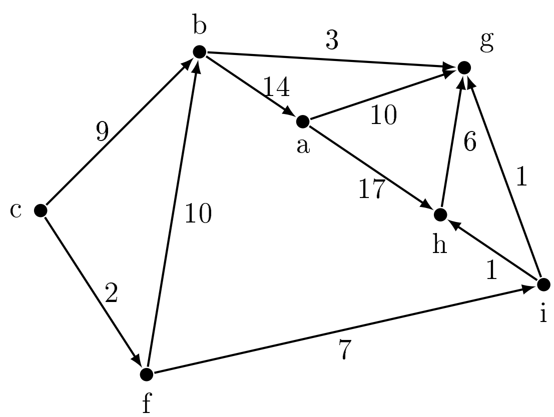
$$\sum_{\text{leafs } u} w(u)\ell(u).$$

The traveling salesman problem For $S = \{\text{Hamilton paths}\}$ minimise $\sum_{e\in H} w(e)$, where $w : E \to R$ is an edge weighting.

2 The shortest path problems

2.1 The directed shortest path problem.

Given a weighted digraph $G = (V, E, \ell)$, where the *positive* weighting $\ell : E \to R_+$ is called *length*, and a specified source $s \in V$ and sink $t \in V$.



1. What is the shortest directed path between c and a?
2. What is the shortest (undirected) oriented path between c and a?

The first problem is to find a *directed shortest path* from s to t. We can also try to find a maximal distance-minimising rooted (at s) directed tree in G — a distance tree —, i.e. a maximal sub-digraph T so that all branches out from s are shortest paths that minimise distance, i.e. for all u in $V(T)$ the path in T from s to u is a shortest path.

1. What is the distance-tree from c above?
2. Will this be an instance of the MST problem? NO.

3. Why positive lengths? What about negative length cycles.

4. Must $V(T)$ be spanning tree? NO. Describe the cut between $V(T)$ and $V \setminus V(T)$.

As will be explained later in the course, we solve these problem "dually". Instead of concetrating on the distance-tree, we try to construct a "dual" solution, namely the function $L : V \to R$, given by

$$L(v) = \text{dir-dist}_G(s, v),$$

where dir-dist(a, b) is the length of a shortest directed ab-path.

The function L is an example of a *value function*. Note: vertices can be thought of as "states" and $L(v)$ is the *value* for the problem — if we are at state v — to find the shortest anti-directed path to the *goal* s.

We have, for $v \in V$ a kind of recursive formula for L

$$L(v) = \min \{\ell(vu) + L(u) : u \in N_-(v)\}. \tag{*}$$

1. If $P : sx_1 \dots zy$ is a shortest sy-path, is $sx_1 \dots z$ a shortest sz-path? Yes

2. Show that (**??**) determines L uniquely, i.e. if $L(v)$ is any function satisfying (**??**), together with the boundary condition $L(s) = 0$, then $L(v)$ must be the sought value function dir-dist$_G(s, v)$. (This is a special case of Bellmans optimality principle.)

2.2 The main loop in Dijkstra's algorithm

Algorithm 1 Main loop of value iteration.

procedure DIJKSTRA(G, ℓ) ▷ Digraph G and $\ell : E(G) \to R_+$
    For $v \in V$, let $L(v) \leftarrow \infty$ if $v \neq s$ and $L(s) \leftarrow 0$ and $P(v) \leftarrow \emptyset$.
    while $\exists x \, \exists y \in N_-(x) \quad L(x) > L(y) + \ell(yx)$ **do**
        $L(x) \leftarrow L(y) + \ell(yx)$
        $P(x) \leftarrow y$.
    end while
    return (L,P)
end procedure

1. Will this always *converge*? Yes — Value improvement $L(u) \searrow$. Will it *stop* in a finite time? It is not immediate at least.
2. How do we recreate the distance-tree from $P(v)$? What is the interpretation of $P(v)$? ($P(v)$ is the parent in the tree.)

2.3 "Safe values" and Dijkstra's algorithm

When can we decide that a value $L(v)$ is safe, i.e. decidedly equal to the distance to s? Initially, we have the safe set $S = \{s\}$, but what about other times?

1. If S is a set of "safe values" at a point in time, i.e. the labeling $L(v)$ is the shortest distance from s. If $x \notin S$. If for all $x \in N_+(S) \setminus S$,

   $$L(x) \leq \min\{\ell(yx) + L(y) : y \in N_-(x) \cap S\}$$

   show that we can extend S, i.e. there is some $v \notin S$ such that $L(v)$ is the right value.

The consideration of safe values give the following refinement of Dijkstra's

Algorithm 2 The safe version of Dijkstra's algorithm

procedure DIJKSTRA(G, ℓ)
    Initialise $L(v)$ and $P(v)$.
    $S \leftarrow \{s\}$.
    while $N_+(S) \setminus S \neq \emptyset$ **do**
        for $x \in V \setminus S$ **do**
            for $y \in N_-(x) \cap S$ **do**
                if $L(x) > L(y) + \ell(yx)$ **then**
                    $L(x) \leftarrow L(y)0\ell(yx)$.
                    $P(x) \leftarrow y$.
                end if
            end for
        end for
        Let x minimise for $L(x)$, $x \notin S$ and let $S \leftarrow S \cup \{x\}$
    end while
    return (L, P)
end procedure

After each iteration of the loop in **??**, we have that

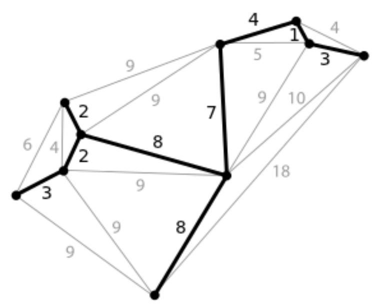
$$L(x) = \min\{\ell(yx) + L(y) : y \in N_-(x)\}.$$

Thus it is safe to extend S with one more element.

1. What does the condition $S \neq N_+(S)$ mean? How to change this if we only want a shortest st-path?
2. What about complexity? As it stands $O(|V| \times |E|)$.
3. Improvements? Do not scan all vertices in $V \setminus S$. Keep the set $N_+(S)$ in a heap ordered by L. ...

3 The minimum weight spanning tree problem

Given a edge-weighted connected graph $G = (V, E, w)$, the minimum spanning tree (MST) or minimum weight spanning tree is a spanning tree with weight less than or equal to the weight of every other spanning tree. More generally, any undirected graph (not necessarily connected) has a minimum spanning forest, which is a union of minimum spanning trees for its connected components.



An example would be a phone company laying cable to a new neighborhood and it is constrained to bury the cable only along certain paths, then there would be a graph representing which points are connected by those paths. Some of those paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights.

While MST are quite easy to find, the minimum spanning tree has a cousin which is algorithmically hard to solve. In the general *Steiner tree problem*

(Steiner tree in graphs), we are given an edge-weighted graph $G = (V, E, w)$ and a subset $S \subset V$ of required vertices. A Steiner tree is a tree in G that spans all vertices of S. In the optimization problem associated with Steiner trees, the task is to find a minimum-weight Steiner tree, but this optimization problem is *NP-hard*.

3.1 The greedy tree and Kruskal's algorithm

Recall the general greedy (tree) forest algorithm: It takes a graph G with a prescribed edge-ordering; a bijection $\pi : [1, m] \to E$, and returns the spanning forest.

```
procedure Greedy(G = (V, E, π))                ▷ Graph G with E ordered
    Initialise tree T = (V, ∅)
    for e ∈ E in the order π do
        if T + e has no cycle then
            T ← T + e
        end if
    end for
    return T
end procedure
```

Kruskal's algorithm takes a (multi-) graph G and constructs a greedy tree in the order of increasing weight. That is.

```
procedure KruskalMST(G = (V, E, w))            ▷ Graph G with E ordered
    Let π order the edges increasing weight.
    return GreedyForest(G, π)
end procedure
```

1. Prove that if e is an edge of minimum weight $w(e)$ in G then there is some MST T containing E. (We can exchange $T' \leftarrow T + e - e'$, so that $w(T') \leq w(T)$, with equality if and only if there is a cycle where all edges have minimum weight.)

2. Use this to prove that Kruskal's algorithm is correct. (Hint: Induction on G/e.)

3. Show that the MST is unique if all edge-weights are distinct.

4. For the wheel graph W_4, assign the weights $1, 1, 2, 2, 3, 3, 4, 4$ to the edges so that (a) the MST is unique and (b) the MST is non-unique.

5. What if we want to maximise the weight of a tree?

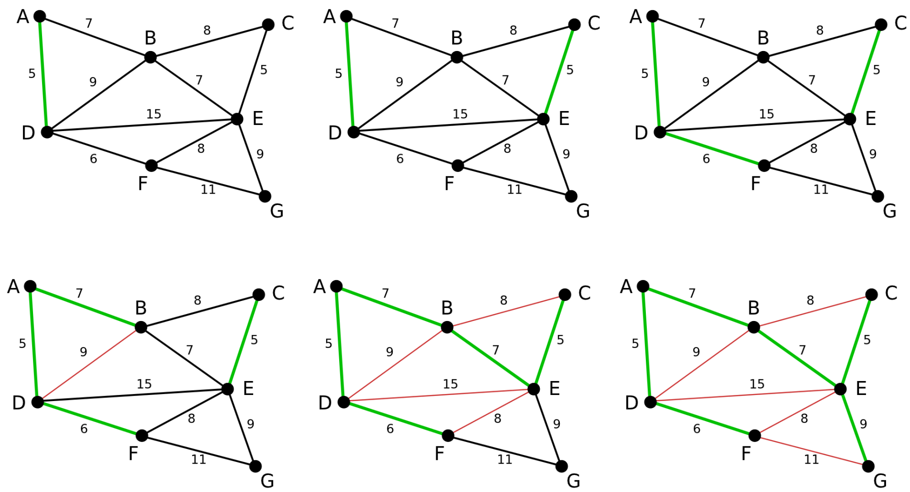


Figure 1: Kruskal's algorithm for a weighted graph

1. What is the complexity of Kruskal's algorithm.

3.2 Boruvka's and Prim's algorithm

Another variant is Prim's algorithm, which has the property that the tree is built up as a growing tree rahter than a growing forest.

procedure PRIM($G = (V, E, w), s$) $\triangleright$ Connected edge-weighted graph with root
  Initialise tree $T = (\{s\}, \emptyset)$
  Let $S \leftarrow \{s\}$
  while $V \setminus S$ is non empty **do**
    Let uv, $u \in S$, $\notin S$ be of minimum weight in $E(S, V \setminus S)$
    $S \leftarrow S + v$.
  end while
  return T
end procedure

A variant of GREEDY is the following which depends on subroutines relative a queue/heap Q and a partition $\mathcal{P} = \{S_1, \ldots, S_r\}$ of $V(G)$.

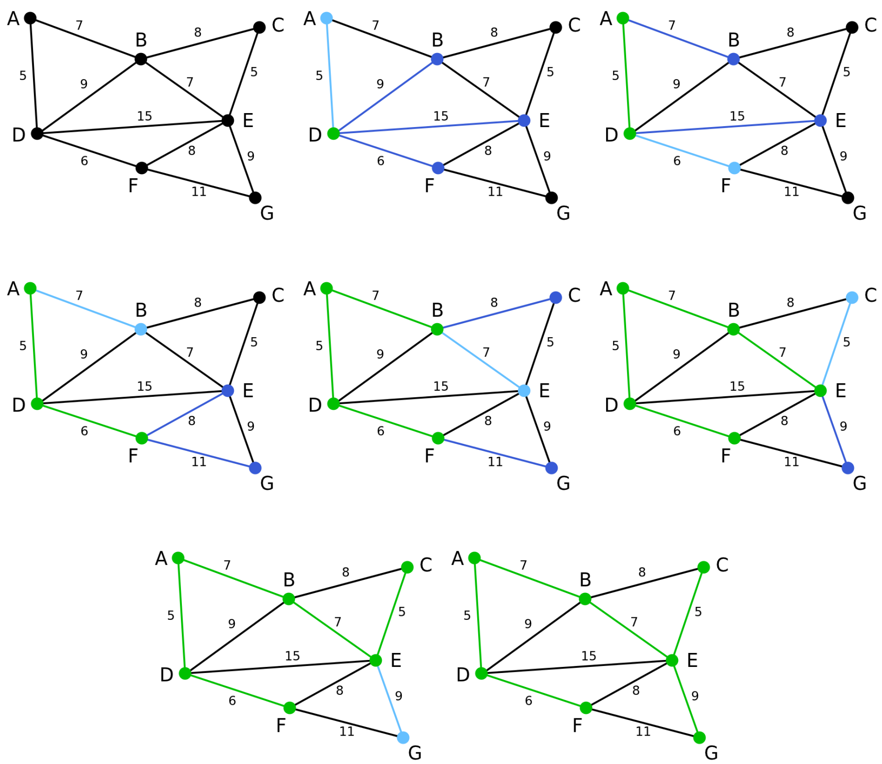


Figure 2: An example of Prim's algorithm for finding an MST in a weighted graph

1. Delete S form Q — remove the elements elements in S from queue (heap) Q.
2. First(Q) Returns the "first" element on Q according to some ordering.
3. A subroutine to obtain the part $\mathcal{P}(u)$ of $\mathcal{P}$ containing u and a subroutine to obtain all edges between to parts $E(\mathcal{P}(u), \mathcal{P}(v))$.

If first gives the element of minimum weight, this is called Boruvka's algorithm for the minimum weight.

procedure Boruvka($G = (V, E, \pi)$) ▷ Graph G with E ordered
  Initialise tree $T = (V, \emptyset)$
  $Q \leftarrow E$
  $\mathcal{P} = \{\{v_1\}, \ldots, \{v_n\}\}$ ▷ Partition into connected components of T

while $uv \leftarrow \textsc{First}(Q)$ **do**
$T \leftarrow T + uv$
$\textsc{Delete}$ edges $E(\mathcal{P}(u), \mathcal{P}(v))$ from Q
$\mathcal{P} \leftarrow \mathcal{P} \setminus \{\mathcal{P}(u), \mathcal{P}(v)\} + \{\mathcal{P}(u) \cup \mathcal{P}(v)\}$.
end while
return T
end procedure

With a smart choice of data structures (a "soft heap") Chazelle obtained an algorithm with complexity $O(|E|\alpha(|V|))$, where α is the inverse of the Ackerman function. ("Almost constant".)