CS 577- Intro to Algorithms

Greed (Part 4)

Dieter van Melkebeek

October 22, 2020

Outline

Discrete multivariate optimization

- System consisting of n components.
- Each component can be in any of a finite number of states.
- Want to set the states of the components so as to optimize an certain objective under certain constraints.

Paradigm

- Consider components in some order.
- Locally optimize setting of each component.

Correctness argument

- Greed stays ahead: interval scheduling, shortest paths
- Exchanges: interval scheduling, minimizing maximum lateness, optimal binary codes

Outline

Discrete multivariate optimization

- System consisting of n components.
- Each component can be in any of a finite number of states.
- Want to set the states of the components so as to optimize an certain objective under certain constraints.

Paradigm

- Consider components in some order.
- Locally optimize setting of each component.

Correctness argument

- Greed stays ahead: interval scheduling, shortest paths
- Exchanges: interval scheduling, minimizing maximum lateness, optimal binary codes, minimum spanning tree

Minimum Spanning Tree

Minimum Spanning Tree

Problem

Input: connected graph $G = (V, E)$ and $w : E \to \mathbb{R}$

Minimum Spanning Tree

Problem

Input: connected graph $G = (V, E)$ and $w : E \to \mathbb{R}$

Ouput: tree $T = (V, F)$ with $F \subseteq E$ such that $w(T) \doteq \sum_{e \in F} w(e)$ is minimized

Minimum Spanning Tree

Problem

Input: connected graph $G = (V, E)$ and $w : E \to \mathbb{R}$

Ouput: tree $T = (V, F)$ with $F \subseteq E$ such that $w(T) \doteq \sum_{e \in F} w(e)$ is minimized

Greedy algorithm

Minimum Spanning Tree

Problem

Input: connected graph $G = (V, E)$ and $w : E \to \mathbb{R}$

Ouput: tree $T = (V, F)$ with $F \subseteq E$ such that $w(T) \doteq \sum_{e \in F} w(e)$ is minimized

Greedy algorithm

- Tree growing
  - Maintain minimum spanning tree T for connected subgraph induced by $S \subseteq V$.
  - Grow S until it reaches V.

Minimum Spanning Tree

Problem

Input: connected graph $G = (V, E)$ and $w : E \to \mathbb{R}$

Ouput: tree $T = (V, F)$ with $F \subseteq E$ such that $w(T) \doteq \sum_{e \in F} w(e)$ is minimized

Greedy algorithm

- Tree growing
  - Maintain minimum spanning tree T for connected subgraph induced by $S \subseteq V$.
  - Grow S until it reaches V.
- Tree joining
  - Maintain minimum spanning forest T of G.
  - Join two trees of the forest until there is only a single tree left.

Tree Growing

Tree Growing

Grow set $S \subseteq V$ and maintain MST T for subgraph induced by S.

- Start with $S = \{s\}$ and trivial MST T.
- While $S \neq V$, find $(u^*, v^*) = \arg\min_{(u,v) \in E \cap S \times \overline{S}} (w(u, v))$ and connect v^* to T via (u^*, v^*).

Tree Growing

Grow set $S \subseteq V$ and maintain MST T for subgraph induced by S.

- Start with $S = \{s\}$ and trivial MST T.
- While $S \neq V$, find $(u^*, v^*) = \arg\min_{(u,v) \in E \cap S \times \overline{S}}(w(u,v))$ and connect v^* to T via (u^*, v^*).

Implementation

Priority queue for $v \in \overline{S}$ with key $\lambda(v) \doteq \min_{u \in S : (u,v) \in E}(w(u,v))$

Tree Growing

Grow set $S \subseteq V$ and maintain MST T for subgraph induced by S.

- Start with $S = \{s\}$ and trivial MST T.
- While $S \neq V$, find $(u^*, v^*) = \arg\min_{(u,v) \in E \cap S \times \overline{S}} (w(u,v))$ and connect v^* to T via (u^*, v^*).

Implementation

Priority queue for $v \in \overline{S}$ with key $\lambda(v) \doteq \min_{u \in S : (u,v) \in E} (w(u,v))$

Running time with binary heap

- Initialization: $O(n)$
- n min extractions: $O(n \log n)$
- $m = \frac{1}{2} \sum_{v \in V} \deg(v)$ key updates: $O(m \log n)$
- Total: $O((n+m) \log n) = O(m \log n)$ as $m \geq n-1$

Tree Joining

Tree Joining

Coarsen partition of G into connected subgraphs, and maintain collection T of MSTs for each of the subgraphs.

- Start with partition consisting of all individual vertices, and trivial spanning forest T.
- While T has more than one subgraph, add (u^*, v^*) to T where $(u^*, v^*) = \arg\min_{(u,v)\in E: u \not\sim_T v}(w(u, v))$.

Tree Joining

Coarsen partition of G into connected subgraphs, and maintain collection T of MSTs for each of the subgraphs.

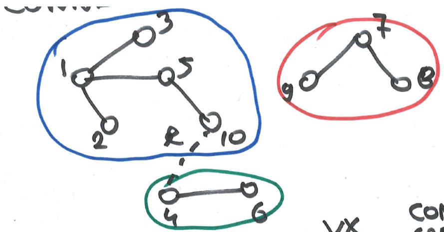
- Start with partition consisting of all individual vertices, and trivial spanning forest T.
- While T has more than one subgraph, add (u^*, v^*) to T where $(u^*, v^*) = \arg\min_{(u,v) \in E : u \not\sim_T v}(w(u, v))$.

Implementation

- Consider edges $(u, v) \in E$ in order of nondecreasing weight.
- Add (u, v) to T if $u \not\sim_T v$.

Maintaining Connected Components

Maintaining Connected Components



CONN COMP	VXS
1	4, 6
2	1, 3, 5, 2, 10
3	8, 7, 9

WHEN JOINING COMPONENTS, RELABEL SMALLER ONE BY THE LARGER ONE

VX	CONN COMP
1	2
2	2
3	2
4	~~1~~ 2
5	2
6	~~1~~ 2
7	3
8	3
9	3
10	2

Maintaining Connected Components

Lazy relabeling

- When merging connected components, relabel smaller one by the larger one.
- Each time vertex gets relabeled, its connected component grows by a factor at least 2 in size.
- Number of times each given vertex gets relabeled $\leq \log n$.

Maintaining Connected Components

Lazy relabeling

- When merging connected components, relabel smaller one by the larger one.
- Each time vertex gets relabeled, its connected component grows by a factor at least 2 in size.
- Number of times each given vertex gets relabeled $\leq \log n$.

Running time

Maintaining Connected Components

Lazy relabeling

- When merging connected components, relabel smaller one by the larger one.
- Each time vertex gets relabeled, its connected component grows by a factor at least 2 in size.
- Number of times each given vertex gets relabeled $\leq \log n$.

Running time

- Sorting the edges: $O(m \log m)$

Maintaining Connected Components

Lazy relabeling

- When merging connected components, relabel smaller one by the larger one.
- Each time vertex gets relabeled, its connected component grows by a factor at least 2 in size.
- Number of times each given vertex gets relabeled $\leq \log n$.

Running time

- Sorting the edges: $O(m \log m)$
- Testing edges: $O(m)$

Maintaining Connected Components

Lazy relabeling

- When merging connected components, relabel smaller one by the larger one.
- Each time vertex gets relabeled, its connected component grows by a factor at least 2 in size.
- Number of times each given vertex gets relabeled $\leq \log n$.

Running time

- Sorting the edges: $O(m \log m)$
- Testing edges: $O(m)$
- Maintaining connected components: $O(n \log n)$

Maintaining Connected Components

Lazy relabeling

- When merging connected components, relabel smaller one by the larger one.
- Each time vertex gets relabeled, its connected component grows by a factor at least 2 in size.
- Number of times each given vertex gets relabeled $\leq \log n$.

Running time

- Sorting the edges: $O(m \log m)$
- Testing edges: $O(m)$
- Maintaining connected components: $O(n \log n)$
- Total: $O(m \log(m) + n \log(n)) = O(m \log n)$ as $m \geq n - 1$

Better Algorithms

Better Algorithms

Based on tree growing

- Binary heap: $O(m \log n)$
- Improved data structures (Fibonacci heaps): $O(m + n \log n)$

Better Algorithms

Based on tree growing

- Binary heap: $O(m \log n)$
- Improved data structures (Fibonacci heaps): $O(m + n \log n)$

Based on tree joining

- $O(m \log m)$ due to sorting edges
- Lazy relabeling: $O(m + n \log n)$ given sorted edges
- Improved data structures (Union-Find): $O(m \cdot \alpha(n, m))$ given sorted edges, where α is inverse Ackermann

Better Algorithms

Based on tree growing

- Binary heap: $O(m \log n)$
- Improved data structures (Fibonacci heaps): $O(m + n \log n)$

Based on tree joining

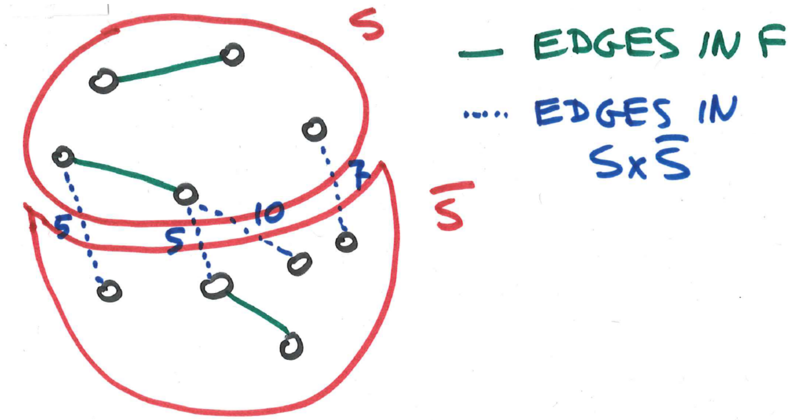
- $O(m \log m)$ due to sorting edges
- Lazy relabeling: $O(m + n \log n)$ given sorted edges
- Improved data structures (Union-Find): $O(m \cdot \alpha(n, m))$ given sorted edges, where α is inverse Ackermann

Other approaches

$O(m \cdot \alpha(n))$ where α is inverse Ackermann

Correctness

Correctness



Correctness

Common setting

- Suppose we know a subset $F \subseteq E$ such that there exists an MST T of G that contains F.
- Consider a subset $S \subseteq V$ such that no edge in F crosses the cut $(S, \overline{S})$, i.e., $F \cap S \times \overline{S} = \emptyset$.

Observation

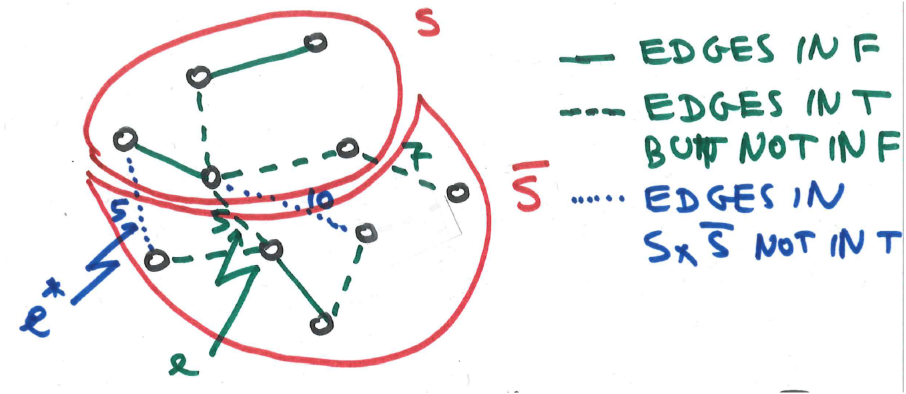
T has to contain an edge in $E \cap S \times \overline{S}$. This edge contributes at least $\min_{e \in E \cap S \times \overline{S}}(w(e))$ to $w(T)$.

Cut property

Let $e^* = \arg\min_{e \in E \cap S \times \overline{S}}(w(e))$.
There exists an MST T' of G that contains $F \cup \{e^*\}$.

Exchange Argument

Exchange Argument



Exchange Argument

Cut property

Let $e^* = \arg\min_{e \in E \cap S \times \overline{S}} (w(e))$.
There exists an MST T' of G that contains $F \cup \{e^*\}$.

Proof

- Suppose e^* not in T; otherwise done.
- Consider adding e^* to T. This induces cycle that crosses $(S, \overline{S})$ somewhere else, say at $e \in E \cap S \times \overline{S}$.
- Replacing e by e^* in T yields spanning tree T' of G.
- Since $w(e^*) \leq w(e)$,

$$w(T') = w(T) + w(e^*) - w(e) \leq w(T).$$

- $\therefore$ T' is an MST of G containing $F \cup \{e^*\}$

Instantiations

Instantiations

- Apply cut property with F the set of edges included thus far.

Instantiations

- Apply cut property with F the set of edges included thus far.
- Tree growing: S is set of vertices in current tree.

Instantiations

- Apply cut property with F the set of edges included thus far.
- Tree growing: S is set of vertices in current tree.
- Tree joining: S is set of vertices connected to u^* in current forest, where (u^*, v^*) is edge under consideration.