Minimum Spanning Tree Problems

AU4606: Network Optimization

Xiaoming Duan
Department of Automation
Shanghai Jiao Tong University

November 27, 2023

Last few lectures

- Global minimum cut problems
  - MA ordering
  - Randomization algorithms

Today

1. Minimum spanning tree problem: formulation
2. Algorithms for minimum spanning tree problems
3. Matroids and greedy algorithms

Today

1. Minimum spanning tree problem: formulation
2. Algorithms for minimum spanning tree problems
3. Matroids and greedy algorithms

What is a minimum spanning tree problem

Minimum spanning tree problem

Given an undirected connected graph $G = (N, A)$ and costs c_{ij} for all arcs $(i, j) \in A$, find a spanning tree $T = (N, A')$ of G such that

$$c(T) = \sum_{(i,j) \in T} c_{ij}$$

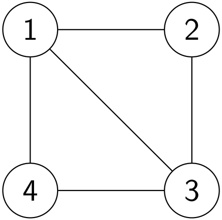
is minimized.

Spanning tree $T = (N, A')$

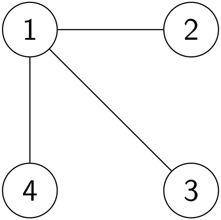
- $A' \subset A$
- T is a connected acyclic graph (tree)

Minimum spanning tree is not necessarily unique (e.g., a graph with uniform arc weights)

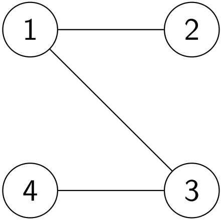
Minimum spanning tree problem: examples



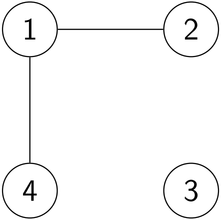
(a) Graph G



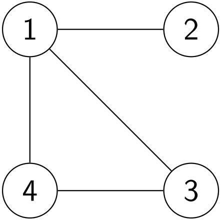
(b) Spanning tree



(c) Spanning tree



(d) Not a spanning tree



(e) Not a spanning tree

Today

1. Minimum spanning tree problem: formulation
2. Algorithms for minimum spanning tree problems
3. Matroids and greedy algorithms

Optimality condition

Cut optimality conditions

A spanning tree T^* is a minimum spanning tree if and only if for every tree arc $(i,j) \in T^*$, $c_{ij} \leq c_{k\ell}$ for every arc (k,ℓ) contained in the cut formed by deleting (i,j) from T^*.

Every arc in an MST is a min cost arc across the cut defined by removing it

Property of minimum spanning tree

Let F be a subset of arcs of some minimum spanning tree. Let S be a set of nodes of some component of F. Then some minimum spanning tree contains all arcs in F and a minimum cost arc $(i,j) \in (S, \bar{S})$.

Optimality condition

Cut optimality conditions

A spanning tree T^* is a minimum spanning tree if and only if for every tree arc $(i, j) \in T^*$, $c_{ij} \leq c_{k\ell}$ for every arc (k, ℓ) contained in the cut formed by deleting (i, j) from T^*.

Every arc in an MST is a min cost arc across the cut defined by removing it

Property of minimum spanning tree

Let F be a subset of arcs of some minimum spanning tree. Let S be a set of nodes of some component of F. Then some minimum spanning tree contains all arcs in F and a minimum cost arc $(i, j) \in (S, \bar{S})$.

Path optimality conditions

A spanning tree T^* is a minimum spanning tree if and only if for every nontree arc $(k, \ell) \notin T^*$, $c_{ij} \leq c_{k\ell}$ for every arc (k, ℓ) contained in the path in T^* connecting nodes k and ℓ.

Kruskal's algorithm: procedure

- Start from a spanning tree, check path optimality condition, swap arcs

Kruskal's algorithm: procedure

- Start from a spanning tree, check path optimality condition, swap arcs

---

Algorithm Kruskal's algorithm

Sort the arcs by their costs $(e_1, e_2, \ldots, e_m)$
$L \leftarrow \emptyset$, $k \leftarrow 1$
while $|L| < n - 1$ **do**
    if Arcs in $L \cup \{e_k\}$ do not form a cycle **then**
        $L \leftarrow L \cup \{e_k\}$
    end if
    $k \leftarrow k + 1$
end while

Correctness of Kruskal's algorithm

Kruskal's algorithm finds a minimum spanning tree for undirected graphs.

Kruskal's algorithm: procedure

- Start from a spanning tree, check path optimality condition, swap arcs

Algorithm Kruskal's algorithm

Sort the arcs by their costs $(e_1, e_2, \ldots, e_m)$
$L \leftarrow \emptyset$, $k \leftarrow 1$
while $|L| < n - 1$ **do**
    if Arcs in $L \cup \{e_k\}$ do not form a cycle **then**
        $L \leftarrow L \cup \{e_k\}$
    end if
    $k \leftarrow k + 1$
end while

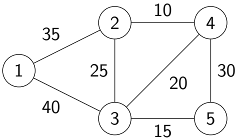
Correctness of Kruskal's algorithm

Kruskal's algorithm finds a minimum spanning tree for undirected graphs.

Detecting cycles

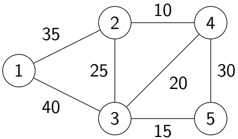
- Maintain a collection of node sets $N_1, N_2, \ldots$
- For (k, ℓ), if both k and ℓ belong to same set, then we have a cycle

Kruskal's algorithm: example

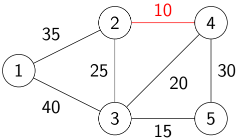


(a) Graph G

Kruskal's algorithm: example

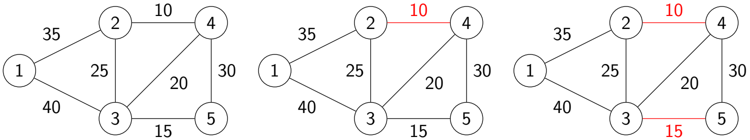


(a) Graph G



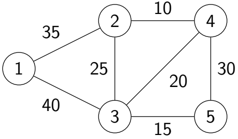
(b) Iteration 1

Kruskal's algorithm: example

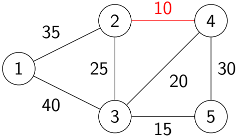


(a) Graph G (b) Iteration 1 (c) Iteration 2

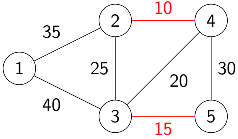
Kruskal's algorithm: example



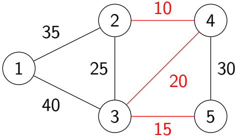
(a) Graph G



(b) Iteration 1

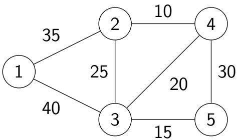


(c) Iteration 2

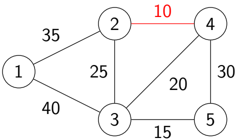


(d) Iteration 3

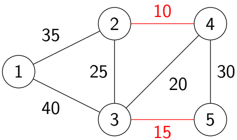
Kruskal's algorithm: example



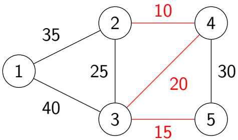
(a) Graph G



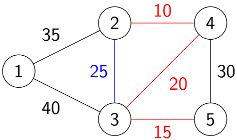
(b) Iteration 1



(c) Iteration 2

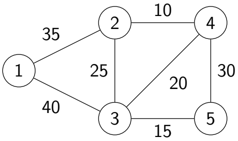


(d) Iteration 3

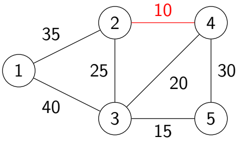


(e) Iteration 4

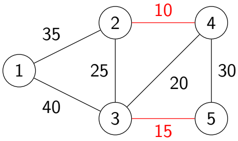
Kruskal's algorithm: example



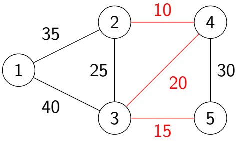
(a) Graph G



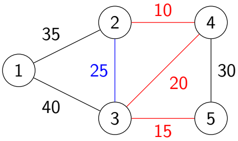
(b) Iteration 1



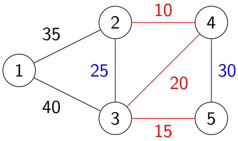
(c) Iteration 2



(d) Iteration 3

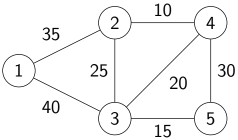


(e) Iteration 4

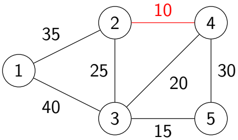


(f) Iteration 5

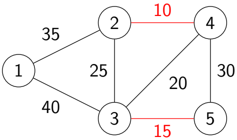
Kruskal's algorithm: example



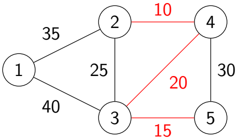
(a) Graph G



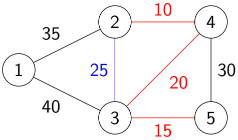
(b) Iteration 1



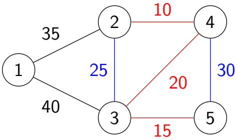
(c) Iteration 2



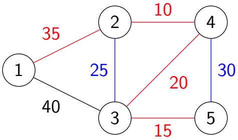
(d) Iteration 3



(e) Iteration 4



(f) Iteration 5



(g) Iteration 6

Prim's algorithm: procedure

Property of minimum spanning tree

Let F be a subset of arcs of some minimum spanning tree. Let S be a set of nodes of some component of F. Then some minimum spanning tree contains all arcs in F and a minimum cost arc $(i, j) \in (S, \bar{S})$.

Prim's algorithm: procedure

Property of minimum spanning tree

Let F be a subset of arcs of some minimum spanning tree. Let S be a set of nodes of some component of F. Then some minimum spanning tree contains all arcs in F and a minimum cost arc $(i,j) \in (S, \bar{S})$.

Algorithm Prim's algorithm

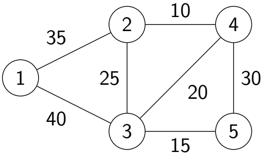
```
1: S ← ∅
2: T ← ∅
3: Add an arbitrary node to S
4: while |S| < n do
5:     (i*, j*) ∈ argmin_{(i,j)∈(S,S̄)} c_ij
6:     S ← S ∪ j*
7:     T ← T ∪ {(i*, j*)}
8: end while
```

- Maintain a tree T on S, add arc with min cost to T from $(S, \bar{S})$

Correctness of Prim's algorithm

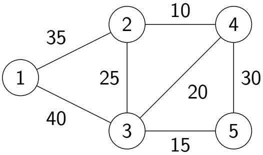
Prim's algorithm finds a minimum spanning tree for undirected graphs.

Prim's algorithm: example

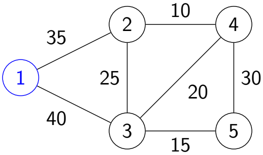


(a) Graph G

Prim's algorithm: example

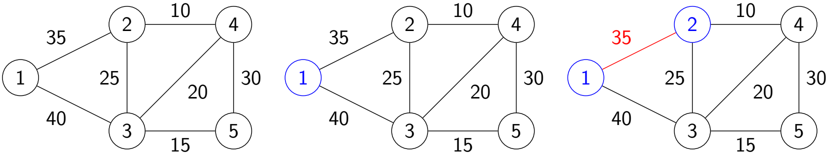


(a) Graph G



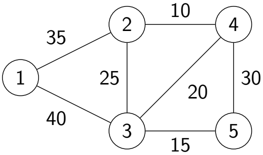
(b) Initialization

Prim's algorithm: example

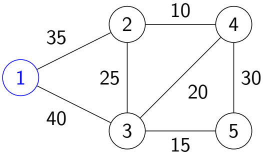


(a) Graph G (b) Initialization (c) Iteration 1

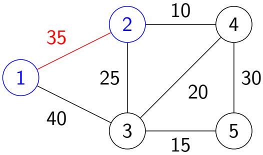
Prim's algorithm: example



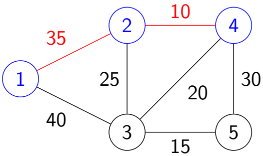
(a) Graph G



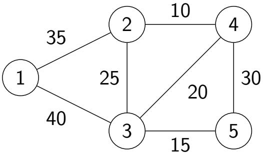
(b) Initialization



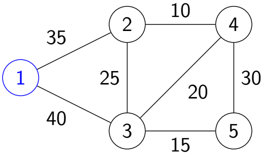
(c) Iteration 1



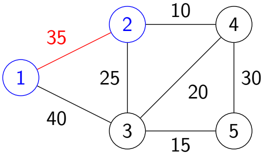
(d) Iteration 2



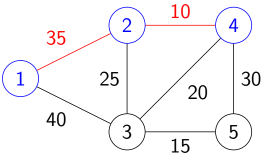
(a) Graph G



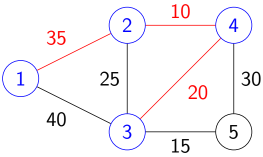
(b) Initialization



(c) Iteration 1

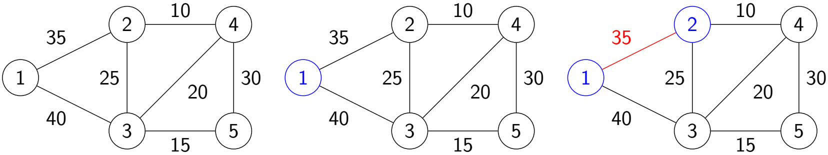


(d) Iteration 2

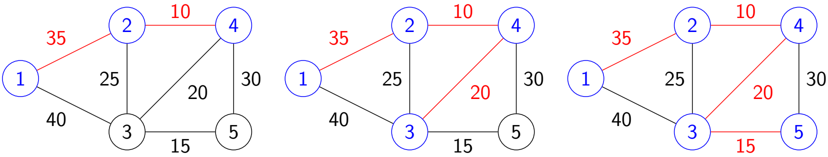


(e) Iteration 3

Prim's algorithm: example



(a) Graph G (b) Initialization (c) Iteration 1



(d) Iteration 2 (e) Iteration 3 (f) Iteration 4

Sollin's algorithm: procedure

Algorithm Sollin's algorithm

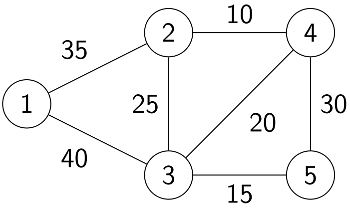
$N_i \leftarrow \{i\}$ for $i \in N$
$T \leftarrow \emptyset$
while $|T| < n - 1$ **do**
  for each N_k **do**
    $(i_k, j_k) \in \arg\min_{(i,j)\in(N_k,\bar{N}_k)} c_{ij}$
  end for
  for each N_k **do**
    if i_k and j_k belong to different trees **then**
      Merge two trees and $T \leftarrow T \cup \{(i_k, j_k)\}$
    end if
  end for
end while

- Maintain multiple trees and merge them until getting a spanning tree

Correctness of Sollin's algorithm

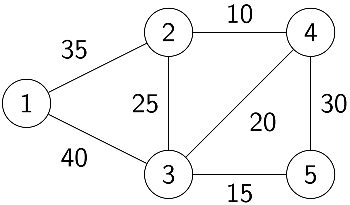
Sollin's algorithm finds a minimum spanning tree for undirected graphs.

Sollin's algorithm: example

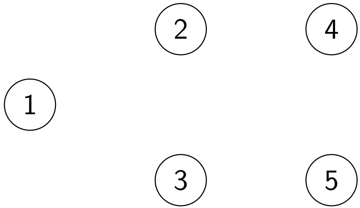


(a) Graph G

Sollin's algorithm: example

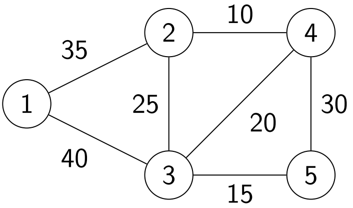


(a) Graph G

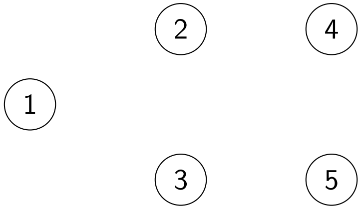


(b) Initialization

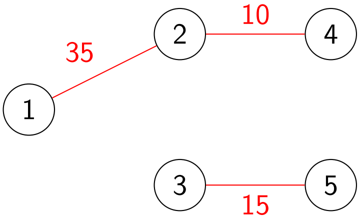
Sollin's algorithm: example



(a) Graph G

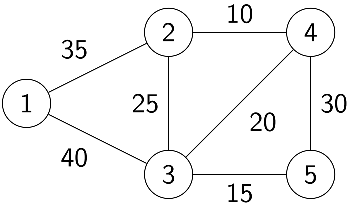


(b) Initialization

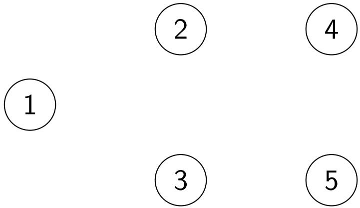


(c) Iteration 1

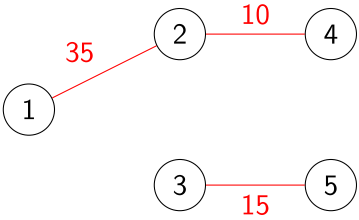
Sollin's algorithm: example



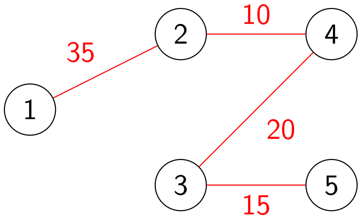
(a) Graph G



(b) Initialization



(c) Iteration 1



(d) Iteration 2

Today

1. Minimum spanning tree problem: formulation
2. Algorithms for minimum spanning tree problems
3. Matroids and greedy algorithms

Kruskal's algorithm is "greedy"

Algorithm Kruskal's algorithm

Sort the arcs by their costs $(e_1, e_2, \dots, e_m)$
$L \leftarrow \emptyset$, $k \leftarrow 1$
while $|L| < n - 1$ **do**
  if Arcs in $L \cup \{e_k\}$ do not form a cycle **then**
    $L \leftarrow L \cup \{e_k\}$
  end if
  $k \leftarrow k + 1$
end while

- Pick an arc with minimum cost at each iteration
- Check if picked arcs constitute a cycle

When does greedy algorithm work?

Matroids

Matroid

A matroid is an ordered pair $(E, \mathcal{I})$ where E is a finite set and $\mathcal{I}$ is a collection of subsets of E that satisfies

1. $\emptyset \in \mathcal{I}$;
2. for any $I_1 \subset I_2$, if $I_2 \in \mathcal{I}$ then $I_1 \in \mathcal{I}$;
3. if $I_1 \in \mathcal{I}$ and $I_2 \in \mathcal{I}$ and $|I_1| < |I_2|$, then there exists $e \in I_2 \setminus I_1$ such that $I_1 \cup e \in \mathcal{I}$.

- The sets in $\mathcal{I}$ are called independent sets
- A maximal independent set is an independent set with max cadinality

Examples of matroids:

- Matric matroid: let M be a real-valued matrix
  1. E is the set of columns of M
  2. $\mathcal{I}$ is the collection of subsets of linearly independent columns
- Graphic matroid: let G be an undirected graph
  1. E is the set of arcs
  2. $\mathcal{I}$ is the collection of subsets of arcs that contain no cycle (forest)

Greedy works for matroid optimization

An optimization problem over matroids

Let $(E, \mathcal{I})$ be a matroid and $w : E \to \mathbb{R}$ be a function that assigns a cost to each element of E. Define $w(X) = \sum_{x \in X} w(x)$ for any nonempty subset $X \subset E$.

Find a maximal independent set that has the minimum cost.

Algorithm Greedy algorithm

1: Sort the elements of $E = \{e_1, \ldots, e_n\}$ so that $w(e_1) \leq \cdots \leq w(e_n)$
2: $L \leftarrow \emptyset$,
3: **for** $i = 1 : N$ **do**
4:     **if** $L \cup \{e_i\}$ is independent **then**
5:         $L \leftarrow L \cup \{e_i\}$
6:     **end if**
7: **end for**

Greedy works

If $(E, \mathcal{I})$ is a matroid, then greedy algorithm finds a maximal independent set that has the minimum cost.

Greedy works for matroid optimization

Greedy works

If $(E, \mathcal{I})$ is a matroid, then greedy algorithm finds a maximal independent set that has the minimum cost.

Greedy works for matroid optimization

Greedy works

If $(E, \mathcal{I})$ is a matroid, then greedy algorithm finds a maximal independent set that has the minimum cost.

Greedy and only greedy works

Let $\mathcal{I}$ be a collection of subsets of a set E. Then $(E, \mathcal{I})$ is a matroid if and only if $\mathcal{I}$ has the following properties

1. $\emptyset \in \mathcal{I}$;
2. If $I \in \mathcal{I}$ and $I' \subset I$, then $I' \in \mathcal{I}$;
3. For all weight functions $w : E \to \mathbb{R}$, the greedy algorithm finds a maximal independent set that has the minimum cost.

A unit-time task scheduling problem

Problem statement

Given

1. a set $S = \{a_1, \ldots, a_n\}$ of n unit-time tasks;
2. a set of n integer deadlines $d(a_1), \ldots, d(a_n)$;
3. a set of nonnegative penalties $w(a_1), \ldots, w(a_n)$;

Find a schedule for S that minimizes total penalties of missed deadlines.

Given a schedule, a task is *early* if finished before ddl and *late* otherwise

Canonical form a task schedule

An arbitrary task schedule can be transformed into the canonical form where early tasks precede late tasks.

A unit-time task scheduling problem

Problem statement

Given

1. a set $S = \{a_1, \ldots, a_n\}$ of n unit-time tasks;
2. a set of n integer deadlines $d(a_1), \ldots, d(a_n)$;
3. a set of nonnegative penalties $w(a_1), \ldots, w(a_n)$;

Find a schedule for S that minimizes total penalties of missed deadlines.

Given a schedule, a task is *early* if finished before ddl and *late* otherwise

Canonical form a task schedule

An arbitrary task schedule can be transformed into the canonical form where early tasks precede late tasks.

Search for a task schedule reduces to search for a set of early tasks!

A unit-time task scheduling problem

Independent set

A set of tasks is independent if there exists a schedule such that these tasks are no late.

A unit-time task scheduling problem

Independent set

A set of tasks is independent if there exists a schedule such that these tasks are no late.

Independence can be checked efficiently

A set of tasks A is independent if and only if $N_t(A) \leq t$ where $N_t(A)$ is the number of tasks whose deadline is t or earlier for $t = 0, 1, \ldots, n$.

A unit-time task scheduling problem

Independent set

A set of tasks is independent if there exists a schedule such that these tasks are no late.

Independence can be checked efficiently

A set of tasks A is independent if and only if $N_t(A) \leq t$ where $N_t(A)$ is the number of tasks whose deadline is t or earlier for $t = 0, 1, \ldots, n$.

Matroid of task scheduling

If S is a set of unit-time tasks with deadlines, and $\mathcal{I}$ is the set of all independent sets of tasks, then $(S, \mathcal{I})$ is a matroid.

A unit-time task scheduling problem

Minimizing penalty of late tasks = maximizing penalty of early ones!

Algorithm Greedy algorithm

```
Sort the tasks S = {a_1, ..., a_n} so that w(a_1) ≥ ··· ≥ w(a_n)
T ← ∅,
for i = 1 : N do
    if T ∪ {a_i} is independent then
        T ← T ∪ {a_i}
    end if
end for
```

Upcoming

Week 1-8 (AU4606 & AI4702):

- Introduction (1 lecture)
- Preparations (3 lectures)
  - basics of graph theory
  - algorithm complexity and data structure
  - graph search algorithm
- Shortest path problems (3 lectures)
- Maximum flow problems (5 lectures)
- Minimum cost flow problems (3 lectures)
- Introduction to multi-agent systems (1 lecture)
- Introduction to cloud networks (1 lecture)

Week 9-16 (AU4606):

- Simplex and network simplex methods (1 lecture)
- Global minimum cut problems (1.5 lectures)
- Minimum spanning tree problems (1.5 lectures)
- Submodular function optimization (2 lectures)
- Optimal assignments and matching (2 lectures)