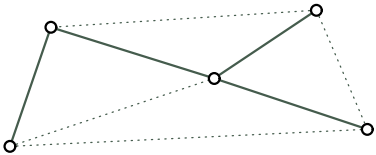
6 MST and Greedy Algorithms

One of the traditional and practically motivated problems of discrete optimization asks for a "minimal interconnection" of a given set of terminals (meaning that every pair will be connected via some path). Imagine, for instance, electric powerline or computer networks.



This problem can be formally captured as finding the minimal connected subgraph of a given (weighted) graph — a minimum spanning tree of the graph.

Brief outline of this lecture

- Minimum spanning tree (MST) problem; Kruskal's greedy algorithm and Jarník's algorithm.
- Principles of greedy algorithms – examples, and when greedy solutions do not work optimally.
- Matroids and abstract greedy optimization.

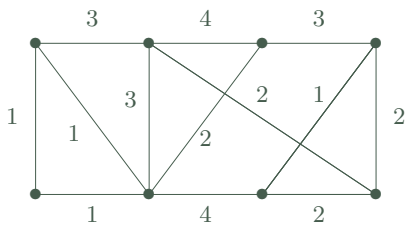
6.1 Finding minimum spanning trees

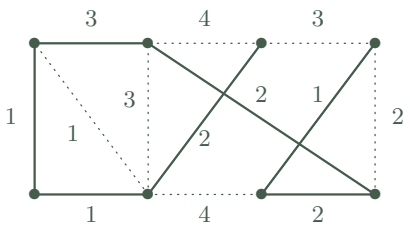
Problem 6.1. Minimum spanning tree (MST) problem
Given a weighted graph G, w with nonnegative edge weights w; the problem is to find a spanning tree T in G that minimizes the total weight. Formally

$$MST = \min_{\text{sp. tree } T \subset G} \left(\sum_{e \in E(T)} w(e) \right).$$

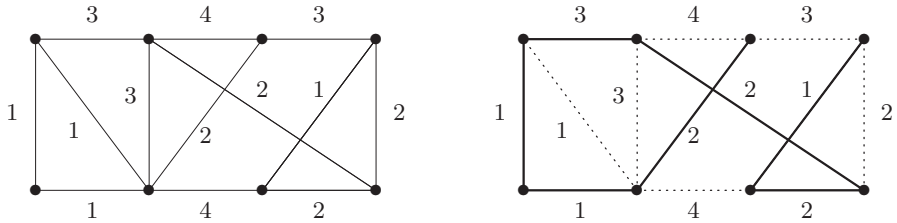
□

In this problem, the spanning tree T is a minimal interconnection with respect to the cost function w. Such as...





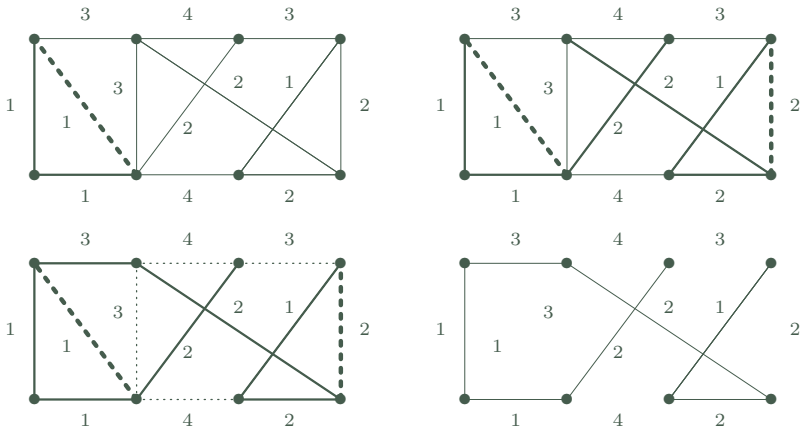
An obvious approach



Algorithm 6.2. "Greedy" algorithm for the MST problem.
Given is a weighted graph G, w with nonnegative edge weights w.

- Sort the edges of G according to nondecreasing weights, i.e. $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$. □
- Start with an empty edge set $E(T) = \emptyset$ for the spanning tree.
- For $i = 1, 2, \ldots, m$, consider the edge e_i: If $E(T) \cup \{e_i\}$ does not make a cycle in G, then add e_i to $E(T)$. Throw e_i "away" otherwise. □
- At the end, $E(T)$ is the edge set of a min. spanning tree T in weighted G, w.

We illustrate the algorithm on our example graph.
(With edge weights sorted as $1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4 \ldots$)



The resultion MST has total weight $1 + 2 + 2 + 3 + 1 + 1 + 2 = 12$.

Notice that the solution (a spanning tree) is not unique, there could be several spanning trees of the same minimum total weight!

Proof of Algorithm 6.2:

Let $E(T)$ be the edge set computed in Algorithm 6.2 and let the edges be sorted as $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$. Among all minimum–weight spanning trees in G, w, we select one T_0 which is identical to T on the longest possible prefix of the edge ordering $e_1, e_2, \ldots, e_m$. If $T_0 = T$, then we are done. □

So assume, for a contradiction, that $T_0 \neq T$. Let $j > 0$ be such that T_0 and T agree on the first $j - 1$ edges $e_1, \ldots, e_{j-1}$, but they disagree on e_j. This means $e_j \in T$ but $e_j \notin T_0$, since the converse cannot obviously happen.

By Corollary 5.5, the subgraph induced on the edges $E(T_0) \cup \{e_j\}$ contains exactly one cycle C. Since $C \not\subseteq T$, there exists an edge e_k of C such that $e_k \notin E(T)$, and $k > j$ by our choice of j. Then $w(e_k) \geq w(e_j)$, and the spanning tree on the edges $\big(E(T_0) \setminus \{e_k\}\big) \cup \{e_j\}$ ("replace" e_k with e_j) costs no more than T_0, and hence it should have been chosen in place of T_0, a contradiction. □

Other possible algorithms

Though the above basic greedy algorithm have been firstly explicitly formulated by Kruskal, other greedy approaches to MST have been found before, noticeably, first by Czech mathematicians. We briefly mention them here:

Algorithm 6.3. Jarník's greedy algorithm for MST.
The edges are not sorted globally in advance, but the spanning tree "grows" from one vertex, at every step choosing the least edge leaving the current fragment of the spanning tree. □

Remark: This algorithm is very practical, and perhaps mostly used in practice. Not many users, however, know its origin and attribute this algorithm to Prim who rediscovered it 30 years after Jarník.

Algorithm 6.4. Borůvka's algorithm for MST (a sketch).
This is a more complicated algorithm which applies the greedy approach "in parallel" from all vertices of the graph at once...

6.2 General greedy algorithms

Perhaps the simplest method of "solving" discrete optimization problems can be summarized as follows:

Always choose what is currently (i.e. locally) the best available local solution.

(Hoping that this will eventually lead to a global optimum.) □

This approach is generally called a *greedy algorithm*. Its core attributes are

- In succesive steps, choose the locally best element as the next one for the solution.
- This approach requires a suitable ordering on the elements (not necessarily determined in advance). □
- The run and the success of such a greedy algorithm strongly depend on this chosen ordering.

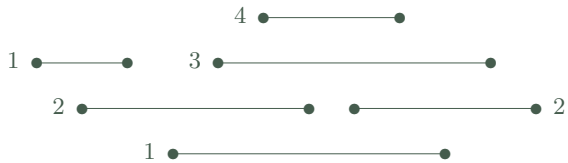
Although greediness is not always polite and successful in real life, surprisingly this approach works great for many problems of discrete optimization, like for the MST problem or others...

Problem 6.5. A job assignment problem

Assume a given list of jobs, each one having specified release and end time (i.e. represented by intervals on the time axis). The task is to assign workers to these jobs such that the total number of workers is minimized. All the workers and jobs are uniform.

□

An example of the input for this problem, see the following:



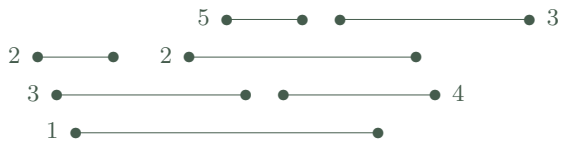
How many workers are needed to satisfy all these jobs? □Quite easily, 4 are enough (see the labeling), but why all 4 are required?

Algorithm 6.6. Greedy algorithm for the job assignment problem.
Problem 6.5 can be solved by the following greedy assignment:

1. *We sort the jobs by their release times.*
2. *For every subsequent job, we assign the least available worker.* □

Proof: Let this algorithm use k workers in total. We easily prove that no less than k workers suffice. So, at what moment the worker of number k started to work? At that moment when all workers $1, 2, \ldots, k-1$ have been busy with other jobs. Hence k jobs overlap at that moment, proving the required. □ □

Is it that we can always assign greedily, regardless of what job ordering we choose? What if we sort the jobs by their lengths (from the longest one)?



As we can see, we need 5 workers in such a case!

6.3 The notion of a matroid

Definition 6.7. A matroid on a ground set X, denoted by $M = (X, \mathcal{N})$, is a set system $\mathcal{N}$ of subsets of X, satisfying the following three points:

1. $\emptyset \in \mathcal{N}$
2. $A \in \mathcal{N}$ and $B \subset A \;\Rightarrow\; B \in \mathcal{N}$
3. $A, B \in \mathcal{N}$ and $|A| < |B| \;\Rightarrow\; \exists y \in B \setminus A : A \cup \{y\} \in \mathcal{N}$

The sets from $\mathcal{N}$ are *independent sets*. The others are *dependent*.
The inclusion-wise maximal independent sets are called *bases* of the matroid. □

The most important in the matroid definition is point three. A natural example of a matroid is given by the linearly independent sets of vectors. □

Lemma 6.8. *All matroid bases have the same cardinality.*

Cycle matroid of a graph

Another natural example of matroids comes from graphs and their spanning trees:

Definition: An edge subset $F \subset E(G)$ is *acyclic* if the graph formed by $V(G)$ and the edges from F contains no cycle.

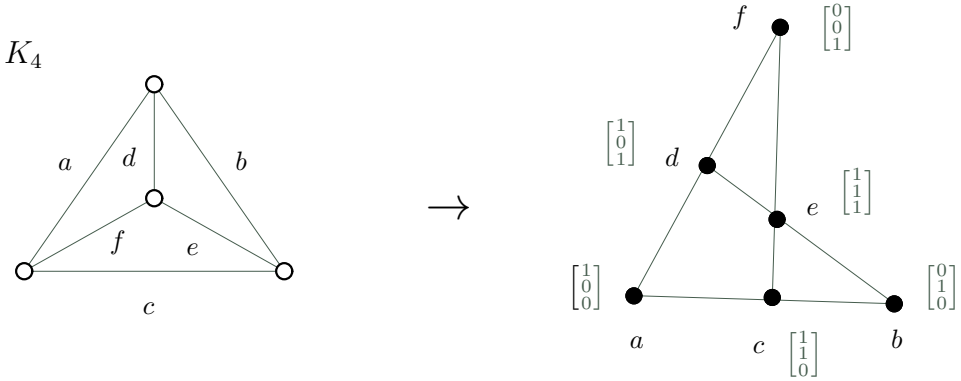
Lemma 6.9. *A forest on n vertices with c components has exactly $n - c$ edges.* □

Lemma 6.10. *Let F_1, F_2 be acyclic subsets of edges in a graph G such that $|F_1| < |F_2|$. Then there exists an edge $f \in F_2 \setminus F_1$ such that $F_1 \cup \{f\}$ is also acyclic.* □

Definition: By Lemma 6.10, the system of all acyclic edge subsets in an arbitrary graph G forms a matroid. This is called the *cycle matroid* of the graph G.

The cycles of G form the minimal dependent sets of this matroid.

How the cycle matroid of a graph corresponds to a vector matroid...



Abstract greedy algorithm

Notice that, concerning algorithmic use of matroids, giving the whole matroid (i.e. listing all its expon. many independent sets) on the input is infeasible. Hence a matroid is commonly described via an external function testing, say, the independent sets. □

Algorithm 6.11. Finding the least independent set – greedily.
`input` $<$ *a set* X *with a weight function* $w : X \to \boldsymbol{R}$*,*
a matroid on X *determined via an external function* `independent(Y);` □

```
sort X=(x[1],x[2],...,x[n]) such that w[x[1]]<=...<=w[x[n]];
B = ∅;
for (i=1; i<=n; i++)
    if (independent(B∪{x[i]}))
        B = B∪{x[i]};
```

`output` $>$ *a basis* B *with the least sum of* w*-weights.* □

Theorem 6.12. *Algorithm 6.11 (the greedy algorithm) on a given ground set* X *and weight function* $w : X \to \boldsymbol{R}$*, and for a given matroid* $\mathcal{N}$ *on* X*, correctly finds a basis* B *of* $\mathcal{N}$ *of the least weight.*

6.4 On (in)correctness of greedy algorithms

On the most abstract level, correctness of the greedy algorithm is tied with matroids as follows.

Theorem 6.13. *Let X be a ground set and $\mathcal{N}$ a system of its subsets such that the items (1),(2) of Definition 6.7 hold. If Algorithm 6.11 correctly finds the optimal independent set from $\mathcal{N}$ for any weight function $w : X \to \boldsymbol{R}$, then also (3) holds for $\mathcal{N}$, and so $\mathcal{N}$ is a matroid on X.* □

Proof by means of a contradiction: If (3) fails for a pair of independent sets A, B, i.e. $|A| < |B|$ but $A \cup \{y\}$ is dependent for any $y \in B \setminus A$, Then we choose a weight function as follows. Let $|A| = a, |B| = b$ where $2b > 2a + 1$, and

- $w(x) = -2b$ for $x \in A$,
- $w(x) = -2a - 1$ for $x \in B \setminus A$,
- $w(x) = 0$ otherwise. □

The greedy algorithm finds a basis $B_1 \supseteq A$, which must be disjoint from $B \setminus A$, and hence of cost $w(B_1) = -2ab$. On the other hand, a basis $B_2 \supseteq B$ has total cost $w(B_2) \leq (-2a - 1)b = -2ab - b < w(B_1)$. This contradicts minimality of B_1 by the greedy algorithm. □

Example 6.14. *Finally, we provide two easy examples of problems for which the greedy approach seriously fails.*

Graph colouring. This problem asks for an assignment of the least number of distinct "colours" to the graph vertices such that adjacent pairs receive distinct colours.□

In a given vertex order, we greedily assign the first available colour to each vertex; see an example:

Are three colours really necessary here? □Of course not...

Vertex cover. Now the task is to find a smallest possible subset C of the vertex set such that every edge has some end in C. □

A natural greedy approach might be to select vertices from the highest degrees, right? □Unfortunately, this procedure again seriously fails in some cases.

□