# Minimum Spanning Tree Optimization Problems and Techniques

Maria Ross

## 1 Introduction

Combinatorial optimization is an area of optimization which is deeply intertwined with combinatorics and graph theory. Combinatorial optimization methods tend to be useful for solving optimization problems involving graphs or other combinatorial structures, so combinatorial optimization methods are often used for problems which can be modeled with such structures. These problems can come from various areas of mathematics, programming, and sciences, and can involve graphs modelling real-world optimization problems or graphs used abstractly in mathematics.

We will explore a few combinatorial optimization algorithms which were created to solve a specific problem: the minimum spanning tree problem. In order to state this problem, we will need to first introduce some definitions and ideas from graph theory.

A (finite) *graph* $G$ is composed of a pair of sets $(V(G), E(G))$, called the *vertex set* and the *edge set* of $G$. The vertex set is a finite set of elements that form or represent the vertices of the graph, and $E(G)$ is a subset of $V(G) \times V(G)$. If $(v_1, v_2) \in E(G)$ for $v_1, v_2 \in V(G)$, we say that $v_1$ and $v_2$ are *adjacent* in $G$. The edge $(v_1, v_2) \in E(G)$ is *incident* to its endpoints $v_1$ and $v_2$.

In a graph $G$, a *path* between $v$ and $u$ in $V(G)$ is an ordered list $\{v_1, v_2, \ldots, v_n\}$ such that $v_1 = v, v_n = u$, and $(v_i, v_{i+1}) \in E(G)$ for $1 \leq i \leq n-1$.

A *tree* is a graph in which there is exactly one path between any two vertices. A *forest* is a collection of trees.

A *subgraph* $H$ of a graph $G$ is a graph such that $V(H) \subseteq V(G)$ and $E(H) \subset E(G)$. If $H$ is a subgraph of $G$, then the *complement of* $H$ is the subgraph $H'$ of $G$ such that $V(H') = V(G)$ and $E(H') = E(G) \setminus E(H)$. A graph is *connected* if there exists at least one path between any two vertices.

Let $G$ be a connected graph. A *spanning tree* for $G$ is a subgraph $T$ of $G$ such that $V(T) = V(G)$

and $T$ is a tree. That is, $T$ is a subgraph which is a tree and includes every vertex of $G$.

We will also need a commonly known lemma regarding spanning trees:

**Lemma 1.** *(From [2], Lemma 2.2) A connected subgraph $H$ of $G$ with $|V(H)| = |G(H)| = n$ is a spanning tree if and only if $H$ has $n-1$ edges.*

If $G$ is a graph, we can consider a *weight function* $\omega : E(G) \to \mathbb{R}$ which assigns weights to the edges of $G$. These weights can represent the lengths of edges, the capacity of edges, or any other numeral value that may be associated with the edges of a graph. We will refer to a graph together with a weight function as a *weighted graph.*

If $G$ is a connected, weighted graph with weight function $\omega$, then a *minimum spanning tree* for $G$ is a spanning tree $T$ such that

$$\sum_{e \in E(T)} \omega(e)$$

is minimized over all possible spanning trees.

Then, the *Minimum Spanning Tree Problem* is the following:

Given a connected graph $G$ with a weight function $\omega$,

$$\text{Minimize} \sum_{e \in E(T)} \omega(e) \qquad \text{over all spanning trees } T.$$

## 1.1 History and Importance

The minimum spanning tree, or MST, problem rose to popularity in the late 1950s with a 1956 paper by Joseph B. Kruskal, Jr titled *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem* (see [4]) and a 1957 paper by R. C. Prim titled *Shortest Connection Networks and Some Generalizations* (see [5]). Kruskal and Prim are commonly viewed as the pioneers of this problem (see [3]), but the problem seemed to have been worked on earlier in Czechia. A Czech mathematician named Otakar Boruvka had previously written a paper stating the ideas of the problem and an algorithm to construct the minimum spanning tree of a graph. Both Kruskal and Prim reference Boruvka's paper, which both refer to as "obscure". Although Boruvka developed the first algorithm for this problem, the algorithms of Kruskal and Prim are much more commonly used and studied (Graham and Hell, p.44).

Boruvka first became aware of the minimum spanning tree problem because he was involved in the electrification of rural parts of Czechia after World War I and was, in his own words, asked to find the most economical construction of an electric power network. He did this by stating and solving the minimum spanning tree problem (Graham and Hell, p.50).

Another Czech mathematician named Vojtech Jarnik first developed the minimum spanning tree algorithm now referred to as "Prim's algorithm" in 1930, nearly three decades before Prim discovered and published this algorithm independently.

Although Czech mathematicians such as Boruvka and Jarnik seem to have developed the first algorithms for this problem, the algorithms of Kruskal and Prim are much more commonly used and studied, and Kruskal and Prim are often credited as the sources of the minimum spanning tree problem.

## 2 Classic Algorithms for Constructing a MST

In his 1956 paper *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*, Kruskal describes three variations of an algorithm to construct the minimum spanning tree for a connected weighted graph $G$ with weight function $\omega : E(G) \to \mathbb{R}$. These variations are as follows:

### 2.1 Kruskal's Algorithm (a).

Begin with subgraph $T$ of $G$ with $E(T) = \emptyset$. While $T$ is not a spanning tree for $G$,

1. Choose $e \in E(G) \setminus E(T)$ such that the subgraph with edge set $E(T) \cup \{e\}$ is a forest (that is, adding $e$ to the subgraph $T$ will not form any cycles) and such that $\omega(e)$ is minimal.

2. Update $T$ by letting $E(T) = E(T) \cup \{e\}$.

Continue this process until $T$ is a spanning tree. Then, $T$ will be a minimum spanning tree for $G$.

Basically, we begin with no edges, and at every step we add the least weight edge such that we create no cycles. We continue until the forest becomes a spanning tree.

### 2.2 Kruskal's Algorithm (b).

Choose a nonempty vertex subset $S \subset V(G)$. Once again, let $T$ be the subgraph of $G$ with $E(T) = \emptyset$. While $T$ is not a spanning tree for $G$,

1. Choose $e \in E(G) \setminus E(T)$ with minimal $\omega(e)$ such that $e$ is either incident to some vertex $v \in S$, or is connected to some edge $e' \in E(T)$, and such that the subgraph with edge set $E(T) \cup \{e\}$ is still a forest.

2. Update $T$ with $E(T) = E(T) \cup \{e\}$.

Continue this process until $T$ is a spanning tree, and once again, $T$ will be a minimum spanning tree for $G$. In the case where $S = V(G)$, it is clear that this algorithm is equivalent to algorithm (a).

The idea is to start with an arbitrary subset of vertices and choose edges with least weight which are incident to vertices in your set. We continue this step until we have a spanning tree.

### 2.3 Kruskal's Algorithm (c).

This variation is complementary to algorithm (a). Kruskal himself refers to it as the 'dual' to algorithm (a). Begin with subgraph $T$ of $G$ with $E(T) = \emptyset$. While the complement $T'$ of $T$ is not a spanning tree for $G$,

1. Choose $e \in E(G) \setminus E(T)$ such that the subgraph with edge set $E(G) \setminus \{e\}$ is still connected and such that $\omega(e)$ is maximal.

2. Update $T$ by letting $E(T) = E(T) \cup \{e\}$.

Continue this process until the no more edges can be added to $T$ while maintaining connectedness. Then the complement $T'$ of $T$ is a spanning tree for $G$ which is minimal.

This algorithm is complementary to Kruskal's algorithm (a) because we begin with the same set and at each step we do an action which is complementary to the action done in algorithm (a). Rather than choosing the least weight edge not in our set which will not form a cycle, we choose the greatest weight edge not in our set which is *not* a cut. Then, we end up with the complement of the minimal spanning tree.

### 2.4 Prim's Algorithm

Prim's algorithm (as stated in [2], p.14) is a simple and common algorithm for constructing minimum spanning trees of graphs. It is as follows:

Choose an arbitrary vertex $r \in V(G)$. Begin with subgraph $T$ such that $V(T) = \{r\}$ and $E(T) = \emptyset$. While $T$ is not a spanning tree,

1. Choose $e \in E(G) \setminus E(T)$ such that $T$ remains a tree and such that $\omega(e)$ is minimal.

2. Update $T$: Let $E(T) = E(T) \cup \{e\}$, and add vertices incident to $e$ to the vertex set $V(T)$.

When $T$ is a spanning tree, $T$ will be a minimum spanning tree of $G$. We note that this algorithm is equivalent to the case of Kruskal's algorithm (b) where the initial vertex subset $S$ is chosen to be a singleton $\{r\}$.

### 2.5 Kruskal's Algorithm, Version 2.

This is really a reformulation of Kruskal's algorithm (a), as stated in [2]. For a connected, weighted graph $G$ with weight function $\omega$ and edge set $E(G)$ with $|E(G)| = m$, write the edges of $E(G)$ ordered as $e_1, e_2, \ldots, e_m$ such that $\omega(e_1) \leq \omega(e_2) \leq \cdots \leq \omega(e_m)$. Start with subgraph $T$ of $G$ with $V(T) = V(G)$ and $E(T) = \emptyset$. For $1 \leq i \leq m$,

1. If $e_i$ is incident to vertices in different components of $T$, then update $T$ by letting $E(T) = E(T) \cup \{e_i\}$.

We can see that this is essentially the same as Kruskal's algorithm (a), except we order the edges beforehand.

## 3 Linear Programs Associated with MST Problems

We can represent the minimum spanning tree problem as a linear program, and we can also use the minimum spanning tree algorithms to solve linear programs of a particular form.

**Theorem 3.1.** *(From [2], Theorem 2.8)*

*If a linear program can be written in the form*

$$\begin{aligned} \textit{Minimize} \quad & \mathbf{c}^T\mathbf{x} \\ \textit{subject to} \quad & \sum_{e \in E} x_e = |V| - 1 \\ & \sum_{e \in E_S} x_e \leq |S| - 1 \textit{ for all } \emptyset \neq S \subset V \\ & x_e \geq 0 \textit{ for all } e \in E \end{aligned} \tag{1}$$

*where $V$ is a finite set, $E \subseteq V \times V$, $E_S = E \cap (S \times S)$, and $\mathbf{x}$ is a vector indexed by the elements of $E$, then we can solve the linear program by finding a minimum spanning tree. In particular, if $T$ is a* *minimum spanning tree for the graph with vertex set $V$ and edge set $E$, then the vector $\mathbf{x}^*$ defined by*

$$x_e^* = \begin{cases} 1 & \textit{if } e \in E(T) \\ 0 & \textit{else} \end{cases}$$

*is an optimal solution to the linear program.*

*Proof.* [2] **Step 1: Rewrite program.** First, we write the program in the form

$$\begin{aligned} \text{Maximize} \quad & -\mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \sum_{e\in E} x_e = |V| - 1 \\ & \sum_{e\in A} x_e \le |V| - c(A) \text{ for all } \emptyset \ne A \subset E \\ & x_e \ge 0 \text{ for all } e \in E \end{aligned} \tag{2}$$

where $V$ is a finite set, $E \subseteq V \times V$, $c(A)$ denotes the number of connected components of the subgraph $G_A$ with edge set $A$, and $\mathbf{x}$ is a vector indexed by the elements of $E$.

First, we show that satisfaction of the constraints in the original program (1) implies satisfaction of the constraints in the rewritten program (2):

For each $A \subset E$, let $\ell = c(A)$ and let $S_1, S_2, \ldots, S_\ell$ denote the connected components of $G_A$, and note that these sets form a partition of $V = V(G)$. We assume satisfaction of the constraints from the original program (1) with $E = E(G_A) = A$. Let $A_i = A \cap (S_i \times S_i)$ for $1 \le i \le \ell$, so that

$$\begin{aligned} \sum_{e\in A} x_e &= \sum_{i=1}^{\ell} \sum_{e\in A_i} x_e \\ &\le \sum_{i=1}^{\ell} (|S_i| - 1) \qquad \text{by constraint satisfaction in (ref)} \\ &= |V| - \ell, \end{aligned}$$

so for each $A \subset E$ we have satisfied the corresponding constraint in the rewritten problem (2). This implies that the feasible region of (1) is a subset of the feasible region of (2).

Conversely, suppose the constraints from (2) hold for all $A \subset E$. Then, for each $\emptyset \ne S \subset V$, let $A \subset E$ be such that $A = E_S = E \cap (S \times S)$. Then the number of connected components $c(A)$ is at least

$|V|-|S|+1$, so we have that

$$\begin{aligned}\sum_{e\in E_S} x_e = \sum_{e\in A} x_e &\\ &\leq |V| - (|V|-|S|+1)\\ &= |S|-1,\end{aligned}$$

so we see that the feasible regions of the two programs are equal, and they have equivalent objective functions, so the programs are equivalent.

**Step 2: Dual program.** Next, we consider the dual of the program as written in (2). We note that the total number of constraints is the number of subsets of $|E|$, $2^{|E|}$. So we will have $2^{|E|}$ dual variables, each associated with some subset of $A \subset E$, or with $E$ itself. We denote the dual variable associated with the subset $A \subset E$ by $y_A$. Then, the dual program's objective will be to minimize the function

$$(|V|-1)y_E + \sum_{A\subset E}(|V|-c(A))y_A = \sum_{A\subseteq E}(|V|-c(A))y_A,$$

where we can condense both terms into one sum by letting $A \subseteq E$ (rather than strict containment) since $c(E) = 1$. Then, the dual program is

$$\begin{aligned}\text{Minimize} \quad & \sum_{A\subseteq E}(|V|-c(A))y_A\\ \text{subject to} \quad & \sum_{\substack{A\subseteq E\\ e\in A}} y_A \geq -c_e \quad \text{for each } e\in E\\ & y_A \geq 0 \text{ for } A\subset E, y_E \text{ free.}\end{aligned}$$

We order the edges in $E$ by $e_1, e_2, \ldots, e_m$ from least to greatest weight. We let $T$ denote the spanning tree constructed by Kruskal's algorithm (2.5) and consider $\mathbf{x}^*$ with

$$x_i^* = \begin{cases} 1 & \text{if } e_i \in E(T)\\ 0 & \text{else.}\end{cases}$$

We want to show that $\mathbf{x}^*$ is an optimal solution to our program. The corresponding dual solution is

$\mathbf{y}^*$ given by

$$y_A^* = \begin{cases} c_{e_{i+1}} - c_{e_i} & \text{if } A = E_i \\ -c_{e_m} & \text{if } A = E \\ 0 & \text{else} \end{cases}$$

where $E_i = \{e_1, e_2, \ldots, e_i\}$ for $1 \leq i \leq m-1$.

**Step 3: Feasibility and Complementary Slackness.** We need to show that $\mathbf{x}^*$ is a feasible solution for the primal program. This is clear, because the total number of edges of a spanning tree is $|V| - 1$, and the number of edges in any sub-tree on $|S|$ vertices of a spanning tree is at most $|S| - 1$ (but can be less, since a vertex $v \in S$ may only be adjacent to vertices outside of $S$.) Both of these facts follow from (ref), and imply that all constraints (of (1), which is equivalent to (2)) are satisfied for $\mathbf{x}^*$.

We also need to show that $\mathbf{y}^*$ is a feasible solution to the dual program. We note that for each $e_j \in E$,

$$\begin{aligned} \sum_{\substack{A \subseteq E \\ e \in A}} y_A^* &= y_E^* + \sum_{\substack{A = E_i \\ i \geq j}} y_A^* + \sum_{A \neq \{e_1, \ldots, e_i\}} 0 \\ &= \sum_{i=j}^{m-1} (c_{e_{i+1}} - c_{e_i}) - c_{e_m} \\ &= -c_j. \end{aligned}$$

So not only is $\mathbf{y}^*$ feasible, but it also satisfies the dual constraint at equality for every nonzero entry of $\mathbf{x}^*$.

On the other hand, if $y_A^* > 0$ for some $A$, then we know that $A = E_i$ for some $1 \leq i \leq m$ (where $E_m = E$.) Then, we have

$$\begin{aligned} \sum_{e \in A} x_e &= x_{e_1} + \cdots + x_{e_i} \\ &= |A \cap E(T)| \end{aligned}$$

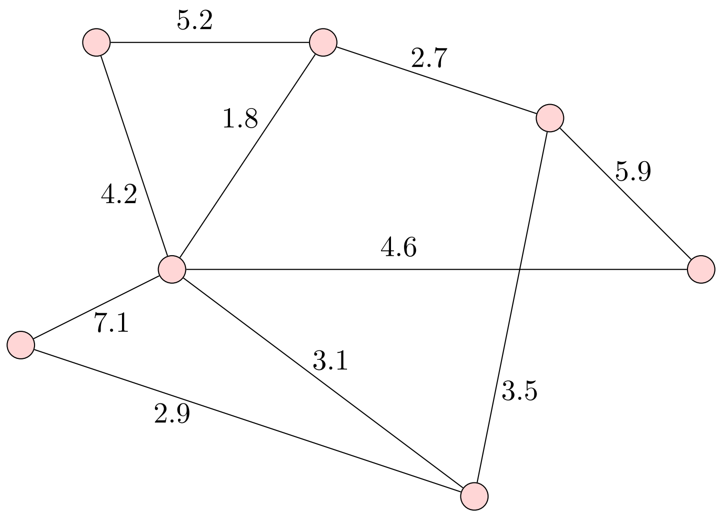
which must be equal to $|V| - c(A)$ because of the construction of $T$ by Kruskal's algorithm. That is, if there is some edge $e \notin A \cap E(T)$ that would decrease the number of graph components by 1, then that edge would have been added to the tree during the algorithm. So, no such edge exists, and complementary slackness is satisfied between primal feasible $\mathbf{x}^*$ and dual feasible $\mathbf{y}^*$. Thus $\mathbf{x}^*$ is an optimal solution to the linear program. □

On the other hand, the previous theorem and proof have made clear that a minimum spanning tree problem can be written as a linear program in this form. For a connected weighted graph $G$ and weight function $\omega$, we let $E = E(G)$ and $V = V(G)$. We use the fact is that any spanning tree of a graph on $n$ vertices must have exactly $n - 1$ edges. Furthermore, we have the property that every vertex must be in the vertex set of the tree, and that we cannot have any cycles in the edge set of the tree. For any vertex subset $S$, we know that a tree restricted to the vertices in $S$ can have at most $|S| - 1$ edges (possibly less, since a vertex in $S$ may only be adjacent in a tree to vertices outside of $S$.) We could also specify that $x_e \in \{0, 1\}$ for each $e \in E$, as our solution vector $\mathbf{x}^*$ will specify which edges are in and not in the spanning tree. Then, it is clear that the problem of finding a minimum spanning tree can be written in the form of the program in Theorem 3.1.

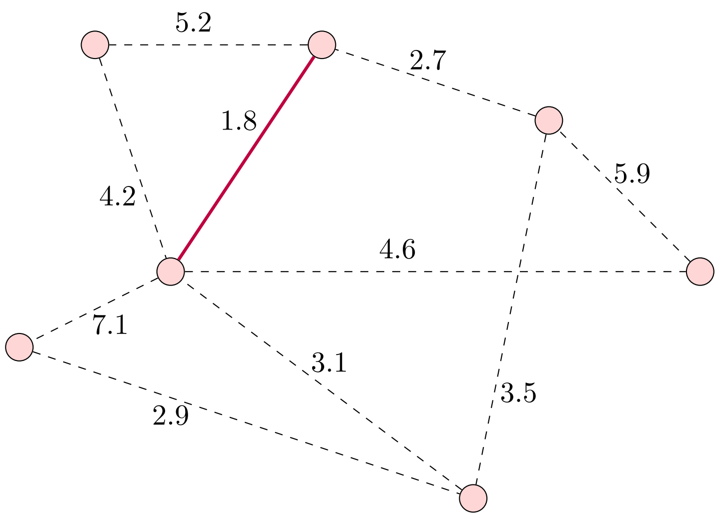
## 4 Examples

### 4.1 Example 1.

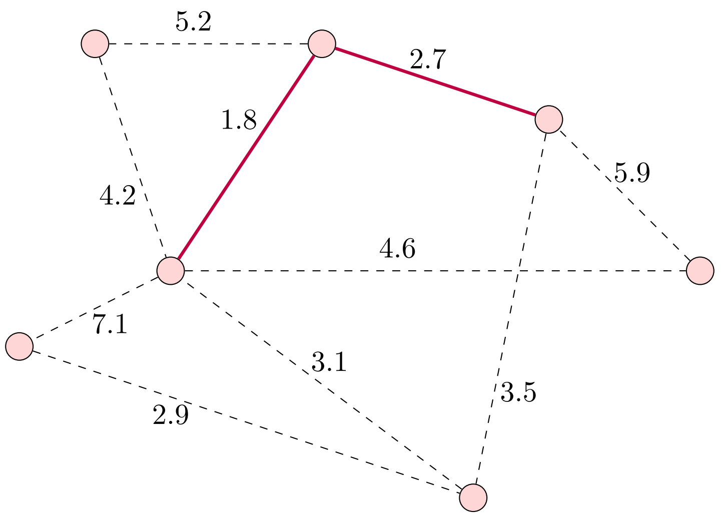
Consider the following graph:

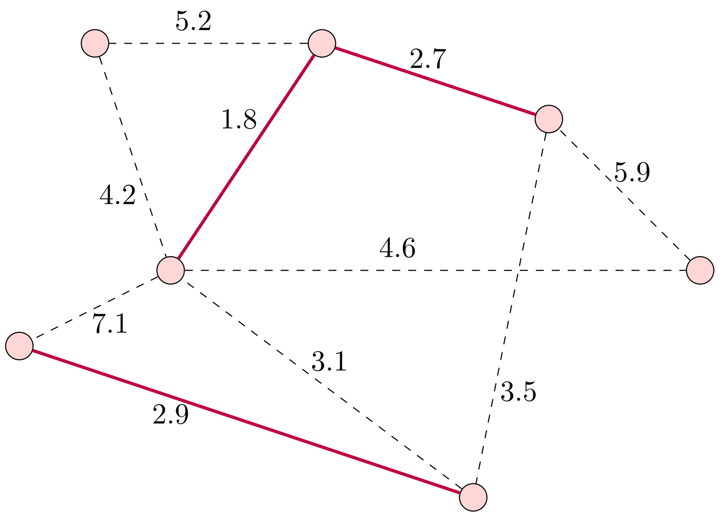


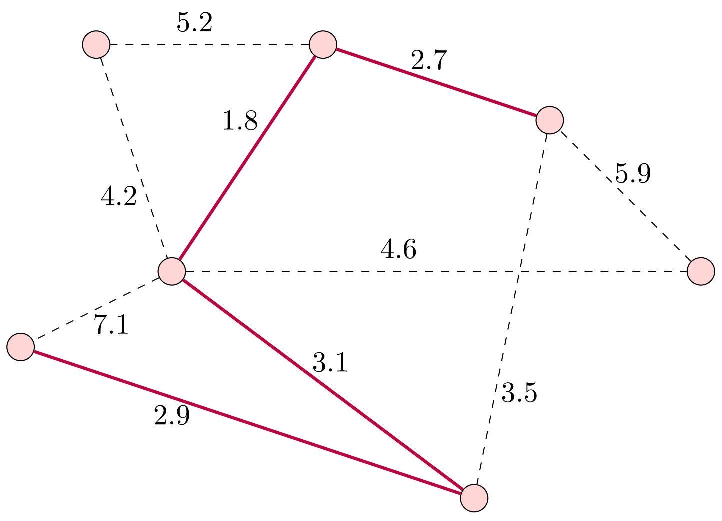
We can use Kruskal's algorithm (2.1) to find the minimum spanning tree for this graph. We start with an empty edge set, and add the edge with the lowest weight:



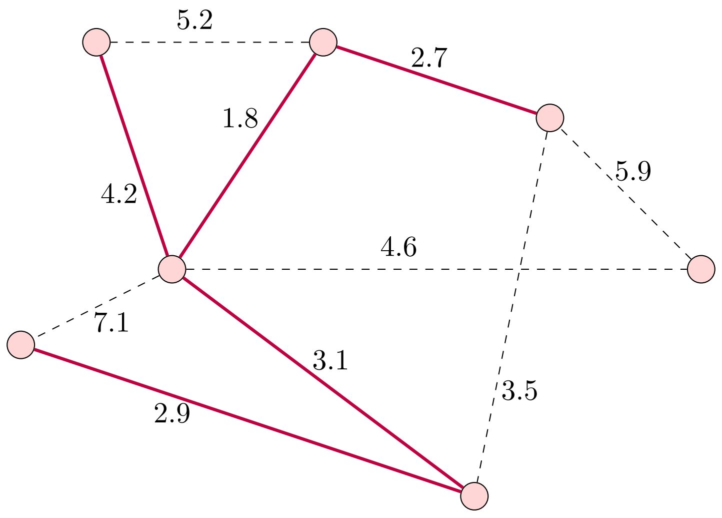
We continue this process:

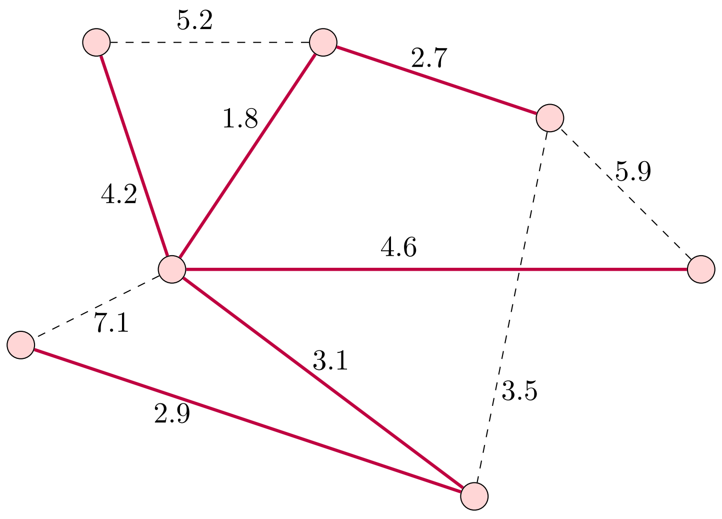






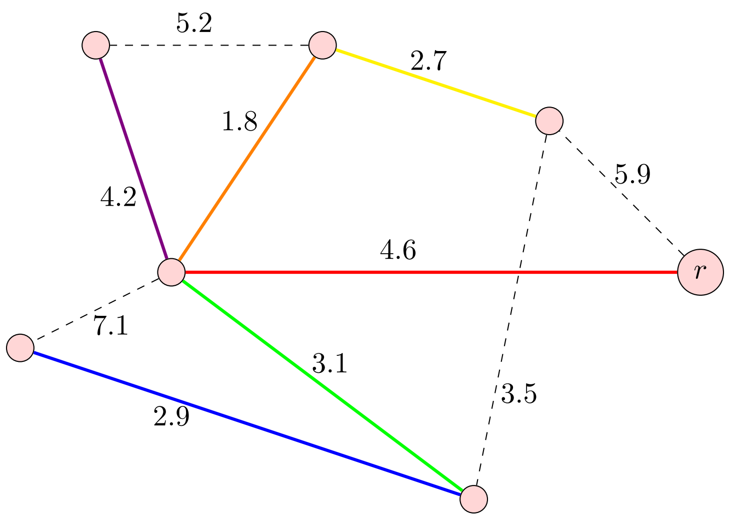
Notice that in the next step, there is an edge with weight 3.5, but adding it to our tree would create a cycle, so we choose the edge with the next smallest weight:





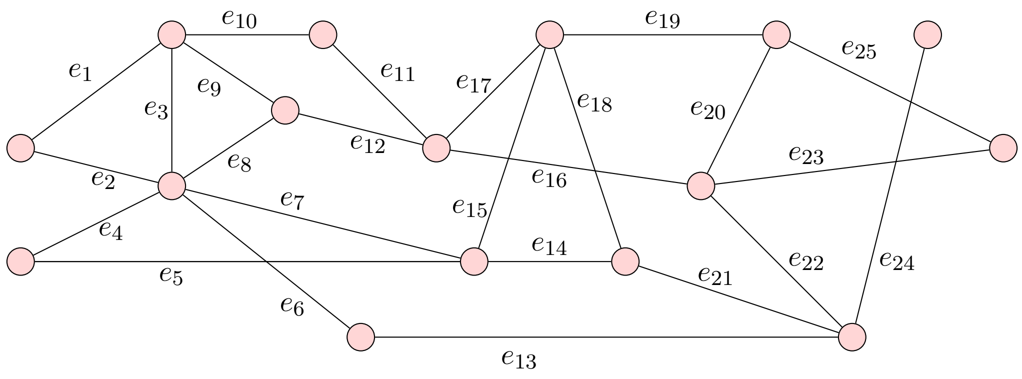
This is our completed minimum spanning tree.

We can also find the minimum spanning tree using Prim's algorithm, choosing the rightmost vertex to be our starting point $r$. All steps are combined into one graph, but the edges would be added in the order of the rainbow.



## 4.2 Example 2.

Consider the following graph $G$:



Also consider the edge weight function $\omega : E(G) \to \mathbb{R}$ defined by

$$\begin{array}{ccccc}
\omega(e_1) = 808 & \omega(e_2) = 577 & \omega(e_3) = 372 & \omega(e_4) = 838 & \omega(e_5) = 691 \\
\omega(e_6) = 366 & \omega(e_7) = 662 & \omega(e_8) = 553 & \omega(e_9) = 663 & \omega(e_{10}) = 897 \\
\omega(e_{11}) = 860 & \omega(e_{12}) = 501 & \omega(e_{13}) = 343 & \omega(e_{14}) = 539 & \omega(e_{15}) = 831 \\
\omega(e_{16}) = 858 & \omega(e_{17}) = 312 & \omega(e_{18}) = 567 & \omega(e_{19}) = 571 & \omega(e_{20}) = 417 \\
\omega(e_{21}) = 796 & \omega(e_{22}) = 448 & \omega(e_{23}) = 749 & \omega(e_{24}) = 503 & \omega(e_{25}) = 624
\end{array}$$

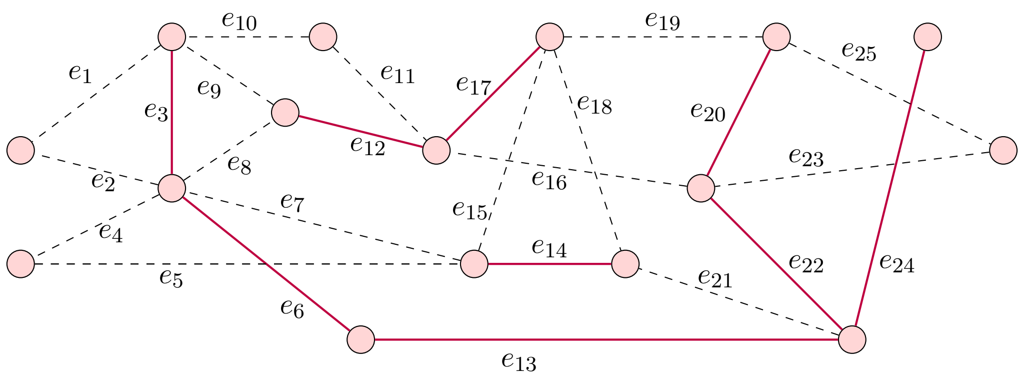
Then, we can use Kruskal's algorithm (2.5): We start by ordering the edges

$$E_{\text{ord}} = (e_{17}, e_{13}, e_6, e_3, e_{20}, e_{22}, e_{12}, e_{24}, e_{14}, e_8, e_{18}, e_{19}, e_2, e_{25}, e_7, e_9, e_5, e_{23}, e_{21}, e_1, e_{15}, e_4, e_{16}, e_{11}, e_{10})$$

We let $T$ be the graph with $V(T) = V(G)$ and $E(T) = \emptyset$, and go through the edges in the order specified above:

1. $e_{17}$ joins two separate components, so $E(T) = \{e_{17}\}$
2. $e_{13}$ joins two separate components, so $E(T) = \{e_{13}, e_{17}\}$
3. $e_6$ joins two separate components, so $E(T) = \{e_6, e_{13}, e_{17}\}$
4. $e_3$ joins two separate components, so $E(T) = \{e_3, e_6, e_{13}, e_{17}\}$
5. $e_{20}$ joins two separate components, so $E(T) = \{e_{20}, e_3, e_6, e_{13}, e_{17}\}$
6. $e_{22}$ joins two separate components, so $E(T) = \{e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$
7. $e_{12}$ joins two separate components, so $E(T) = \{e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$
8. $e_{24}$ joins two separate components, so $E(T) = \{e_{24}, e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$
9. $e_{14}$ joins two separate components, so $E(T) = \{e_{14}, e_{24}, e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$

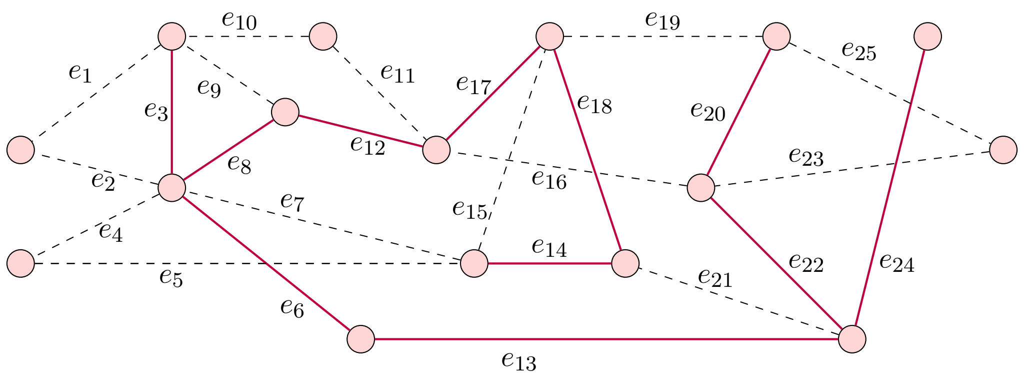
So far, we have



From this point, we will update the graph more often.

10. $e_8$ joins two separate components, so $E(T) = \{e_8, e_{14}, e_{24}, e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$
11. $e_{18}$ joins two separate components, so $E(T) = \{e_{18}, e_8, e_{14}, e_{24}, e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$

Our updated graph:



12. $e_{19}$ does not join two separate components, so we do not add this edge to the tree

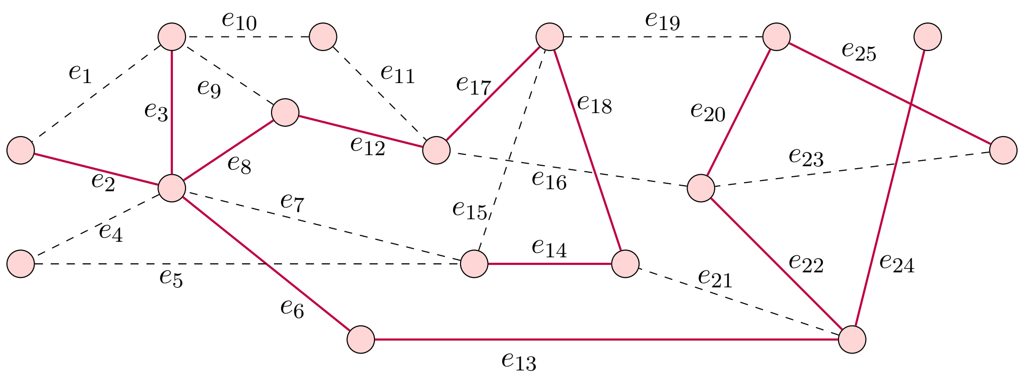
13. $e_2$ joins two separate components, so $E(T) = \{e_2, e_{18}, e_8, e_{14}, e_{24}, e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$

14. $e_{25}$ joins two separate components, so $E(T) = \{e_{25}, e_2, e_{18}, e_8, e_{14}, e_{24}, e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$

15. $e_7$ does not join two separate components, so we do not add this edge

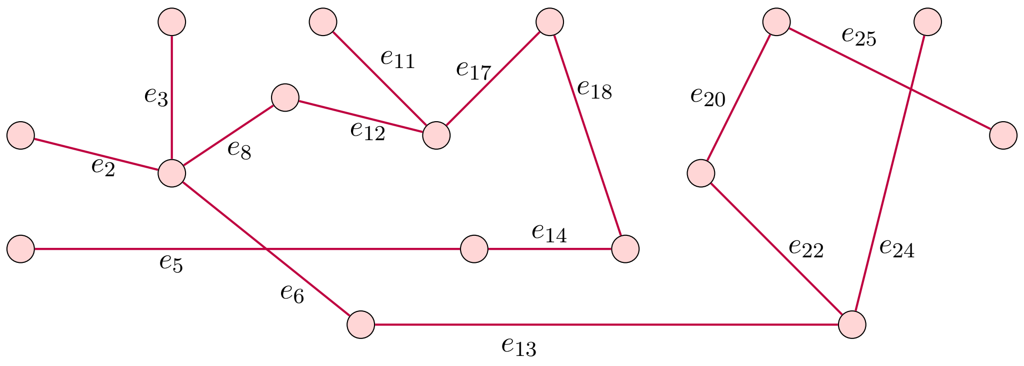
16. $e_9$ does not join two separate components, so we do not add this edge

Our updated graph:



18. $e_5$ joins two separate components, so $E(T) = \{e_5, e_{25}, e_2, e_{18}, e_8, e_{14}, e_{24}, e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$

At this point, we only need one more vertex in the tree. We see that the only two edges which are incident with this vertex are $e_{10}$ and $e_{11}$, and $e_{11}$ comes first in our ordered list, so our final minimum spanning tree has edge set $E(T) = \{e_{11}, e_5, e_{25}, e_2, e_{18}, e_8, e_{14}, e_{24}, e_{12}, e_{22}, e_{20}, e_3, e_6, e_{13}, e_{17}\}$ and can be seen below:



These examples are only abstract, but minimum spanning tree problems can be used to model real-world optimization problems, and are particularly useful for various network problems. Prim writes about practical problems which can be solved using minimum spanning tree algorithms in his paper. The minimum spanning tree problem is also closely related to Steiner's Problem and to the Traveling Salesman Problem, both of which are famous problems with applications in computer science (and combinatorics and discrete mathematics) and in real life situations.

## References

[1] 1. Otakar Boruvka. *On a minimal problem.* Prace Moravske Prirodovedecke Spolecnosti, vol. 3, 1926.

[2] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimizatin.* Chapter 2: Optimal Trees and Paths. Wiley-lnterscience series in discrete mathematics and optimization.

[3] R. L. Graham and Pavol Hell. *On the History of the Minimum Spanning Tree Problem.* Annals of the History of Computing. 7. 43-57.

[4] Joseph B. Kruskal, Jr. *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem.* Proceedings of the American Mathematical Society, Vol. 7, No. 1 (Feb., 1956), pp. 48-50.

[5] Robert C. Prim. *Shortest Connection Networks and Some Generalizations.* Bell System Technical Journal, 36: 6. November 1957 pp 1389-1401.